
Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Francisco Luís Magno Morgado

Concepção de um Pequeno Sensor Inercial 3D

Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Francisco Luís Magno Morgado

Concepção de um Pequeno Sensor Inercial 3D

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Eng. Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. Telmo Reis Cunha, Professor Auxiliar Convidado do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro

o júri / the jury

presidente / president	Prof. Dr. José Carlos Esteves Duarte Pedro Professor Catedrático da Universidade de Aveiro
vogais / examiners commitee	Prof. Dr. Telmo Reis Cunha Professor Auxiliar Convidado da Universidade de Aveiro (orientador)
	Prof. Dr. Jaime dos Santos Cardoso Professor Auxiliar Convidado da Faculdade de Engenharia da Universidade do Porto

Agradecimentos / acknowledgements

Agradeço ao meu orientador por me ter dado a oportunidade de realizar este projecto que faz uso de conceitos e tecnologias de extrema actualidade, interesse e aplicabilidade. Agradeço ao Departamento de Electrónica e Telecomunicações da Universidade de Aveiro pelos recursos disponibilizados e agradeço, finalmente, à minha família e namorada por todo o apoio.

palavras-chave Unidade inercial de medida, MEMS, acelerómetro, giroscópio

resumo Com este trabalho pretende-se desenvolver um IMU (Inertial Measurement Unit), fazendo uso de acelerómetros e giroscópios MEMS (Micro-machined Electro-Mechanical Systems) de baixo custo. Esta tecnologia emergente abre novas portas no mundo dos sistemas de navegação inerciais dada a sua simplicidade mecânica. Este IMU permite transferir as medidas para uma unidade de processamento/armazenamento e, por conseguinte, a sua integração em sistemas de navegação mais complexos que façam uso de multiplas unidades de medida: IMU, GPS - Global Positioning System, odómetro, magnetómetro.

keywords Inertial measurement unit, MEMS, accelerometer, gyroscope

abstract This work aims to develop an IMU (Inertial Measurement Unit), making use of low cost MEMS (Micro-machined Electro-Mechanical Systems) accelerometers and gyroscopes . This emerging technology opens new doors in the world of inertial navigation systems due to its mechanical simplicity. This allows transfer IMU measures to a processing/storage unit, hence its integration into more complex navigation systems that make use of multiple units of measure: IMU, GPS - Global Positioning System, odometer, magnetometers.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Este projecto consiste, essencialmente, em três componentes: a concepção e construção do IMU (Inertial Measurement Unit); o desenvolvimento de um algoritmo para integração dos dados de seis sensores inerciais (três acelerómetros e três giroscópios), para determinação dos deslocamentos linear e angular; e finalmente a análise da qualidade das medidas obtidas.

A primeira componente é essencialmente de desenvolvimento de electrónica. Os acelerómetros e giroscópios que deverão compor o IMU dispõem tipicamente de um sinal em tensão que varia proporcionalmente com a aceleração linear ou velocidade angular respectivamente. Será, portanto, necessário implementar ADCs sincronizadas para converter essa informação no formato digital. A informação em digital deverá ser, então, manipulada por um micro-processador (PIC18 - Microchip), que simultaneamente a enviará por uma porta série RS232. Um sensor adicional de temperatura será também incluído, com o intuito de avaliar a sensibilidade dos sensores às variações da temperatura. Todo o IMU deverá assentar num algoritmo responsável por gerar uma base de tempo que sincronizará todas as medidas. Será feita uma caracterização estatística dos erros produzidos pelos sensores, que será fundamental para os processamentos posteriores, e para a possibilidade futura de integração dos dados deste IMU com outros sensores dedicados à localização (receptores GPS, odómetros, etc.). Finalmente, e mediante vários testes que serão executados, será analisada de uma forma quantificada a qualidade das soluções de navegação (posição e orientação) que este IMU apresenta.

2 Enquadramento

Os sistemas de navegação baseados em sensores de *dead-reckoning* (i.e., cuja informação de navegação se reporta apenas a variações relativas às posições anteriores, e não a posição absoluta actual) têm sido utilizados desde há várias décadas para o posicionamento de sistemas que se deslocam sobre a superfície da Terra (automóveis, navios, etc.), em água (submarinos), no ar (aviões, mísseis) e também no espaço (satélites). Destes sensores, os que se baseam em medir acelerações relativamente a um referencial inercial têm ocupado um lugar predominante - são os elementos de base aos Sistemas de Navegação Inerciais (INS - Inertial Navigation Systems). Com o desenvolver da tecnologia, vários tipos de sensores inerciais foram sendo desenvolvidos, levando à existência de duas classes distintas de sensores - os de elevada precisão (high-grade) e os de menor precisão (low-grade). Naturalmente, os primeiros são extremamente mais caros que os segundos, e são de natureza conceptual muito diferente. Nas duas últimas décadas tem havido uma forte evolução em áreas de investigação que se dedicam à localização robusta de precisão de viaturas que se deslocam sobre a superfície terrestre, com especial ênfase nos automóveis. Com o aparecimento da tecnologia GPS (Global Positioning System), o posicionamento absoluto de precisão tornou-se muito acessível para muitas aplicações. No entanto, quando se considera um automóvel a deslocar-se numa zona urbana, ou em zonas arborizadas, o bloqueio dos sinais GPS provenientes dos satélites torna-se um sério problema que conduz à incapacidade de determinação de posições (ou de posições com a desejada exactidão) em vários locais do trajecto. Para aumentar a robustez destes sistemas de navegação, várias equipas de investigação têm-se dedicado à integração de receptores GPS com sensores de *dead-reckoning* (que não dependem de sinais provenientes do exterior da viatura), sendo frequentemente utilizados sensores inerciais de média e elevada precisão (ver, por exemplo, [4]-[7]). Muitos destes sistemas integrados foram aplicações directas de sistemas integrados desenvolvidos para aeronaves, onde as exigências de precisão são muito elevadas. No entanto, para sistemas terrestres, este rigor posicional é muitas vezes menos exigente, o que possibilita a utilização de equipamento mais barato (mas menos preciso). No caso em concreto dos automóveis, e dada a disponibilidade dos sinais GPS, são pouco frequentes as situações onde o posicionamento final depende apenas dos sensores inerciais durante um largo período de tempo. Mais realista é considerar-se que a informação dos sensores inerciais servem para suavizar os erros associados às posições GPS ou para calcular, durante curtos intervalos de tempo, a posição da viatura quando os sinais GPS são bloqueados. Assim, torna-se interessante testar o comportamento de um sistema de navegação inte-

grado que considere apenas sensores de muito baixo custo (sensores inerciais MEMS), averiguando-se, em situações reais, quais os níveis de precisão e robustez que se atingem com este sistema. Neste sentido, o sistema inercial desenvolvido neste trabalho tem como objectivo ser inserido num sistema de navegação para uma viatura, cujo desenvolvimento dos restantes elementos é considerado nos trabalhos da dissertação de Mestrado Integrado em Engenharia Electrónica e de Telecomunicações, da Universidade de Aveiro, intitulada "Desenvolvimento e análise de desempenho de um sistema de navegação integrado [8].

2.1 Sistema de Navegação Inercial (INS)

Este tipo de sistemas são caracterizados por determinarem a posição processando continuamente medidas que são captadas vindas de sensores inerciais, nomeadamente acelerómetros e giroscópios, dispostos de forma ortogonal. Integrando a informação proveniente dos acelerómetros determina-se a velocidade e, por sua vez, integrando a velocidade determina-se o deslocamento.

A grande vantagem dos INSs advém do facto de não necessitarem de informação proveniente de referências externas. Isto tornou a navegação inercial na solução mais comum para veículos aéreos e marítimos. Mais recentemente, devido aos avanços tecnológicos, o leque de aplicações destes sistemas tem vindo a aumentar e os custos a diminuir, tornando possível, por exemplo, captação da dinâmica do humano.

Tipicamente os INSs inserem-se em uma de duas categorias: nos sistemas de plataforma giro-estabilizada ou nos sistemas fixos (*strapdown*).

Nos sistemas de plataforma giro-estabilizada os sensores inerciais estão montados numa plataforma que procura manter a mesma orientação independentemente de toda a dinâmica a que está sujeita. Isto consegue-se à custa de uma estrutura de três arcos articulados (gimbals) e de três motores que permitem à plataforma total liberdade de orientação - Figura 1. Quando a plataforma é sujeita a um deslocamento angular, os giroscópios quantificam esse movimento fazendo com que os motores contrariem esse deslocamento. Desta forma garante-se que integrando as medidas dos acelerómetros se obtêm deslocamentos lineares relativos ao referencial global.

Nos princípios da década de setenta do século passado, a industria da navegação inercial propunha o desafio de eliminar toda a estrutura de eixos articulados (*gimbals*), que caracterizava os sistemas inerciais da época, fixando os acelerómetros e os giroscópios. Desta forma, processava-se toda a informação sensorial com algoritmos mais poderosos transferindo, assim, toda a complexidade mecânica para os algoritmos matemáticos. Substituíam-

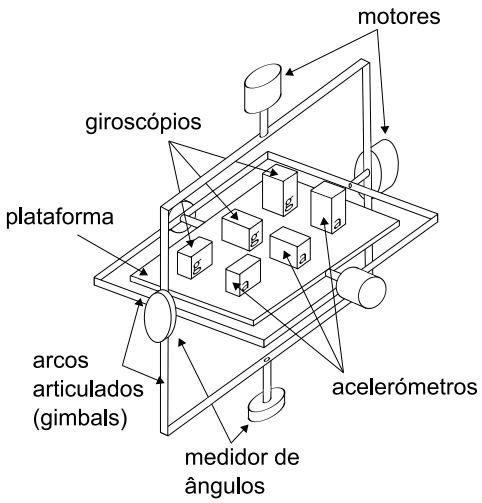

Figura 1: Plataforma giro estabilizada (adaptado de [2])

se os eixos mecânicos articulados por "eixos matemáticos articulados", proporcionando assim o aparecimento dos sistemas inerciais fixos (*strapdown*).

Este tipo de sistemas usa sensores que, ao contrário das plataformas giro-estabilizadas, estão fixados ao corpo cuja dinâmica se pretende descrever, portanto as medidas que daqui são retiradas dizem respeito ao referencial local do corpo. Para que as acelerações provenientes dos três acelerómetros sejam integradas correctamente estas medidas terão de ser previamente projectadas no referencial global, e isso consegue-se à custa de três giroscópios dispostos ortogonalmente que permitem uma contínua estimativa da orientação do referencial local relativamente ao referencial global. Visto isto, eliminam-se os eixos articulados (*gimbals*), e por conseguinte desaparece a necessidade de calibração, tornando estes sistemas mais fiáveis. Por outro lado, esta fixação possibilita que se construam os sistemas em dimensões mais reduzidas e a custo mais baixo. Para além destes benefícios, estes sistemas trazem o inconveniente de serem computacionalmente mais exigentes.

2.2 Giroscópio e Acelerómetro

Nesta secção pretende-se fazer uma breve descrição dos sensores aqui usados referindo alguns aspectos morfológicos e erros mais relevantes que lhes estão associados. Uma descrição mais detalhada pode ser encontrada em [1, 2].

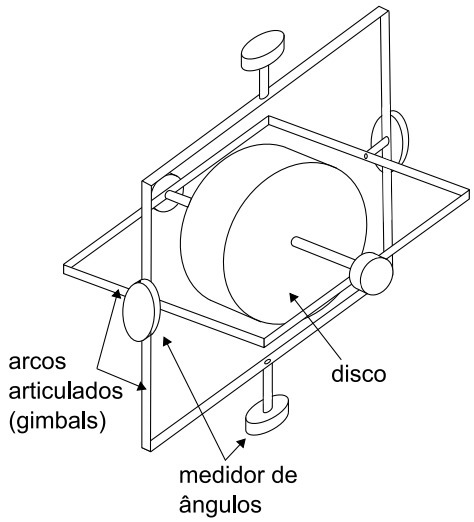

Figura 2: Modelo conceptual de um giroscópio mecânico (adaptado de [2])

2.2.1 Giroscópio MEMS

Giroscópios são sensores com a função de quantificar variações angulares relativamente a um determinado eixo com elevada precisão e ignorar todos os outros tipos de dinâmica (lineares e angulares) - Figura 2. A grandeza dos sinais gerados por estes dispositivos vem tipicamente expressa em °/s e pode atingir muitas centenas de graus por segundo. Há, no entanto, um compromisso entre o alcance das medidas e a precisão.

O princípio que está por trás dos giroscópios MEMS (Micro-machined Electro-Mechanical Systems) é o efeito de Coriolis que estabelece que uma massa M movendo-se com uma velocidade linear v num referencial que roda com uma velocidade angular ω sofre o efeito de uma força

$$F_C = -2 \cdot M \left(\omega \times v \right) \tag{1}$$

O modelo mais simples consiste numa massa de prova que é levada a vibrar ao longo de um eixo X que quando sofre uma rotação em torno de Z induz, segundo o efeito de coriolis, uma vibração no eixo Y, perpendicular a X e Z - figura 3.

Estes sensores são constituídos por um número reduzido de componentes mecânicos, e por conseguinte são relativamente baratos de construir.

As principais vantagens deste tipo de giroscópios relativamente a outros, nomeadamente os mecânicos, são:

- tamanho/peso reduzido

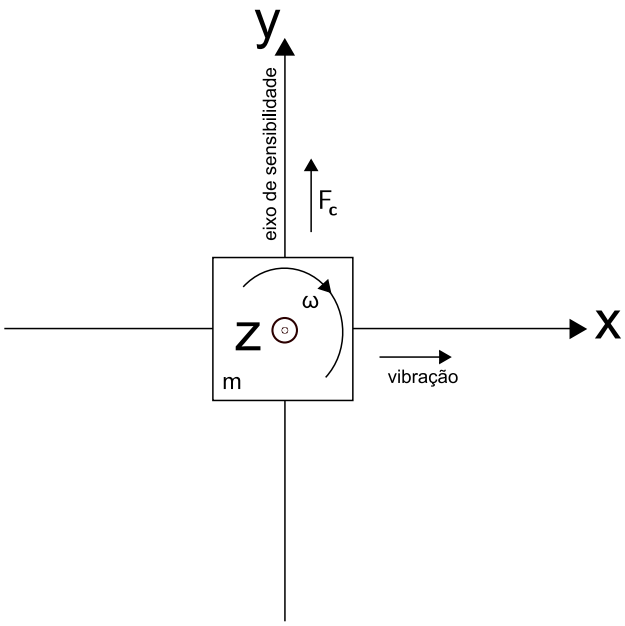

Figura 3: Modelo de um giroscópio MEMS, que faz uso do efeito de coriolis.

- baixo consumo energético
- estrutura sólida
- tempo de start-up reduzido
- custo financeiro reduzido
- grau de fiabilidade elevado
- não requer manutenção
- resistência a ambientes hostis

A grande desvantagem destes giroscópios prende-se com o facto de actualmente ainda não serem tão precisos quanto os giroscópios ópticos [1, 2]. A desvantagem dos giroscópios ópticos é serem várias ordens de grandeza mais caros.

As fontes de erro com maior relevo nos giroscópios MEMS são:

- Erro Sistemático: É possível medir o erro sistemático do sensor integrando as medidas sucessivamente quando não se lhe é imposto qualquer rotação e se verifica uma diferença entre o valor esperado e o valor real calculado em $^\circ/\mathrm{h}$. Uma vez determinado o seu valor ϵ, este pode ser eliminado subtraindo esse valor às medidas

extraídas do sensor. Caso esta compensação não se verifique, os deslocamentos angulares calculados serão afectados de um deslocamento proporcional a ϵ, que cresce linearmente em ordem ao tempo

$$\theta(t) = \epsilon \cdot t \tag{2}$$

- Ruído Branco: As principais causas de ruído branco nos giroscópios MEMS são perturbações termo-mecânicas que geram variações de frequência muito superior à taxa de amostragem. Este ruído é caracterizado por um conjunto de variáveis aleatórias de média nula em que cada uma destas variáveis está igualmente distribuída e caracterizada por uma variância finita σ^2. Desta forma o ruído introduz *random walk* de média nula nas medidas, após integração, com um desvio padrão

$$\sigma_\theta(t) = \sigma \cdot \sqrt{\delta t \cdot t} \tag{3}$$

  Para uma explicação um pouco mais detalhada deve referir-se a [1] e [2].

- Ruído de Baixa Frequência (*Fliker Noise*): O *fliker noise* é responsável por gerar variações que são inversamente proporcionais à frequência (1/f). Este ruído perde relevância para as altas frequências, tornando-se desprezável quando comparado com o ruído branco.

- Efeito de Variações na Temperatura: A relação entre a temperatura e a variação no sinal é altamente não linear. A forma mais comum de eliminar este erro é implementando um sensor de temperatura, cujas medidas são inseridas num modelo matemático de compensação dos parâmetros dos sensores sensíveis às variações da temperatura. Alguns sistemas inerciais contêm internamente sensores de temperatura e por conseguinte fazem uma correcção automática dos erros provenientes desta fonte.

- Erros de Calibração: Os erros de calibração surgem devido ao desalinhamento dos sensores durante a fase de montagem do IMU e aos erros intrínsecos ao acondicionamento dos sinais. A natureza destes erros é muito idêntica à dos erros sistemáticos e reflectem-se em desvios quando integrados.

2.2.2 Acelerómetro MEMS

Os acelerómetros têm por objectivo medir acelerações com alta precisão numa direcção muito bem definida. Estes sinais são tipicamente expressos em m/s^2 ou popularmente

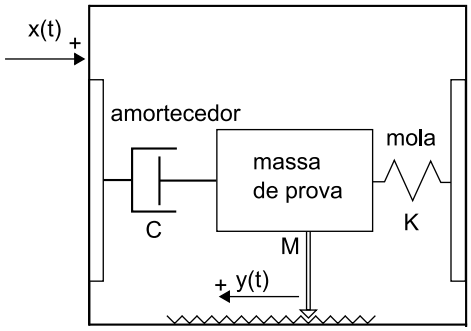

Figura 4: Modelo conceptual do acelerómetro (adaptada de [2]).
Equação da dinâmica: $M\ddot{y} - M\ddot{x} = -Ky - C\dot{y}$

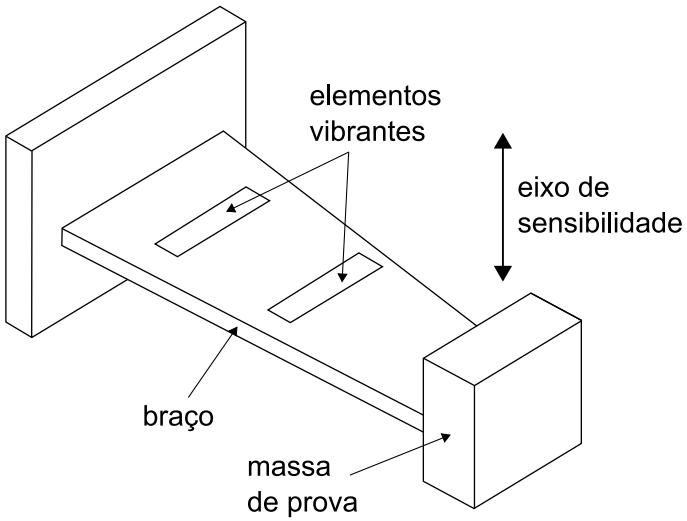

Figura 5: Acelerómetro de tecnologia MEMS.

referida à constante de aceleração gravítica terrestre g. O princípio que está por trás dos acelerómetros MEMS consiste numa massa suspensa por um braço de natureza idêntica ao de uma mola e de um amortecedor em série. Tal como indica a figura 4, fazendo a leitura do deslocamento da massa é possível calcular o valor da aceleração pela segunda lei de Newton

$$F = ma \tag{4}$$

Há ainda outra categoria de acelerómetros MEMS cujo princípio é o de medir as variações da frequência de uma onda acústica de superfície presente num braço flexível com uma massa suspensa - figura 5.

Estes dispositivos usufruem das vantagens da tecnologia MEMS e por isso partilham

das mesmas qualidades previamente mencionadas relativamente aos giroscópios.

Tal como os giroscópios a grande desvantagem dos acelerómetros MEMS é ainda não serem tão precisos quanto os acelerómetros que usam as técnicas tradicionais.

Por partilharem a mesma tecnologia com os giroscópios MEMS acima referidos, os acelerómetros sofrem essencialmente dos mesmos erros, no entanto importa destacar algumas diferenças:

- Erro Sistemático: Quando o acelerómetro está sujeito a condições tais, que o valor do sinal seja previsível, nomeadamente quando está em repouso, é possível determinar o erro sistemático ϵ pela diferença entre o valor real e o esperado que provoca um erro na posição calculada (integrando duplamente a aceleração) que cresce de forma quadrática

$$s\left(t\right) = \epsilon \cdot \frac{t^2}{2} \tag{5}$$

- Ruído Branco: Tal como os giroscópios MEMS os acelerómetros sofrem de perturbações termo-mecânicas geradoras de desvios no sinal. Neste caso, devido à dupla integração do sinal, o ruído branco gera *random walk* de segunda ordem de média nula e desvio padrão

$$\sigma_s \approx \sigma \cdot t^{3/2} \cdot \sqrt{\frac{\delta t}{3}} \tag{6}$$

3 Concepção do IMU

Para a construção do IMU foram tidos em conta essencialmente os seguintes factores: ortogonalidade dos seis sensores inerciais; a redução do número de fontes de erro; e a sua estabilidade e robustez mecânica. Nesta secção será explicado com mais detalhe quais os sensores usados e a forma como foram implementados, tendo em conta os factores previamente mencionados.

No diagrama da figura 6 ilustram-se os andares principais por onde os sinais provenientes dos sensores devem passar até se obter a posição e a orientação do IMU. Este diagrama servirá como guia durante a fase da construção. As acelerações e as velocidades angulares passam pelo andar de amplificação, por forma a ajustar os sinais às características das entradas dos andar seguintes. Este circuito de amplificação é constituido por seis amplificadores concebidos para maximizar a excursão de sinal de cada um dos sinais enviados pelos sensores. Dado que o micro-controlador usado contém apenas um conversor analógico-digital (ADC), com um tempo de conversão obviamente não nulo, seria impossível guardar quaisquer seis medidas que dissessem respeito ao mesmo instante de tempo. Visto isto, implementou-se um andar de sincronização com o objectivo de "memorizar", durante um determinado tempo, as medidas, provenientes do andar de amplificação, permitindo assim à ADC converter os seis sinais relativos ao mesmo instante de tempo. Por outro lado, foi implementado um sensor de temperatura com o intuito de ser feita uma análise dos erros com origem nas variações da temperatura. Depois de toda a informação ser processada, as medidas são enviadas, já em formato digital, para o sistema de armazenamento de dados do sistema de navegação integrado, ou para um computador (para possibilitar vários testes), que fará a integração de todos os dados.

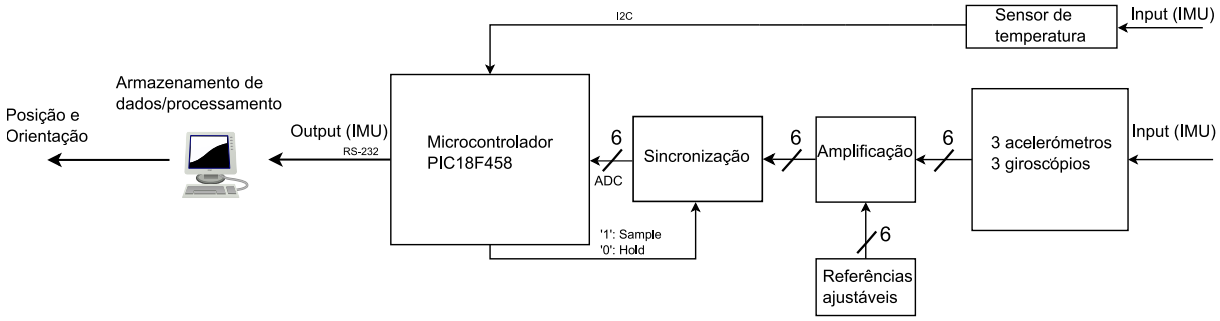

Figura 6: Diagrama do caminho do processamento do sinal

3.1 Escolha dos sensores

Os critérios com maior relevância usados para a determinação dos modelos dos acelerómetros e dos giroscópios a usar foram: o preço, ruído, a sensibilidade às variações de temperatura, o alcance e o tamanho.

Para o acelerómetro, o alcance refere-se aos valores máximo e mínimo para os quais o acelerómetro ainda tem sensibilidade. Este parâmetro vem tipicamente expresso em $g's$ (múltiplos da aceleração da gravidade). Este valor deve ser escolhido de forma a não exceder em demasia as especificações exigidas pela aplicação para que haja um máximo aproveitamento da excursão do sinal. Neste caso, para um implementação onde se consideram condições de circulação normais para uma viatura, espera-se que uma variação até $\pm 0.3g$ seja suficiente. Quanto ao ruído e à sensibilidade devido a variações de temperatura, devem ser tão baixos quanto possível - vêm expressos em mV e em $\%/°C$, respectivamente.

No caso do giroscópio, a mesma ideia aplica-se para o alcance, mas desta vez as unidades vêm expressas em $°/s$. Para esta aplicação não será necessário um intervalo de valores para o alcance superior a $\pm 70°/s$.

Assim sendo, foram escolhidos o acelerómetro ADXL103 (Analog Devices) e o giroscópio MEV-50A-R (Murata), por apresentarem um custo reduzido e por corresponderem às exigências acima mencionadas - figura .

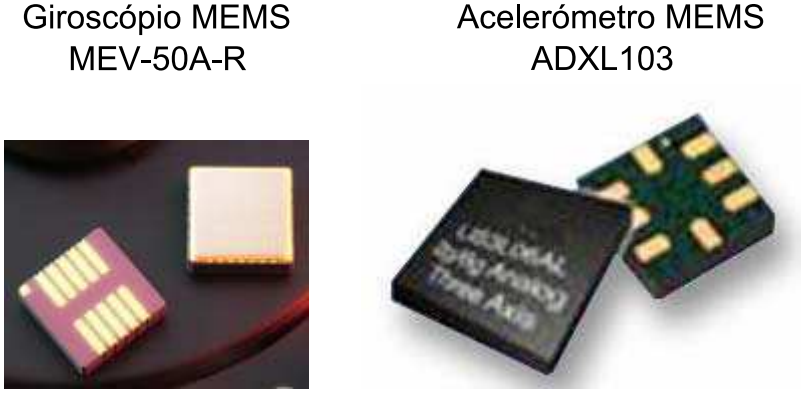

Figura 7: Fotografias dos sensores inerciais usados na concepção do IMU.

Quanto ao sensor de temperatura, foi usado um LM75 (digital) - erro $\pm 1°C$- por não exigir qualquer tipo de calibração. Apesar de ser um pouco mais caro do que outros sensores, tais como a PTC ou o termopar, este apresenta uma razão preço/qualidade melhor.

Na tabela 1 estão descritas as características acima mencionadas para cada sensor.

Sensor	Preço	Alcance	Ruído	Sens. Temp.	Tamanho
ADXL103	$\approx 16\,euro$	$\pm 1,7g$	$6mV\,rms$	$0,3\%/°C$	$5 \times 5 \times 2$
MEV-50A-R	$\approx 14\,euro$	$\pm 70°/s$	$7mV\,rms$	$4\%/°C$	$10 \times 10 \times 3$

Tabela 1: Características dos sensores.

3.2 Projecto do esquemático

No circuito da figura 8, estão projectados os sensores responsáveis por fazer a aquisição dos sinais de entrada (andares de *input* do IMU vistos no diagrama anterior) relativos à dinâmica (acelerações e rotações) e à temperatura . Estão ainda implementados o andar de amplificação e as respectivas referências ajustáveis, que permitirão calibrar o *offset* dos sensores inerciais. O amplificador do giroscópio terá um ganho unitário, já o amplificador do acelerómetro está projectado para um ganho de 8,3. Desta forma, a excursão de sinal que fica acessível à ADC está compreendida entre $[-0.3, 0.3](g)$ ou aproximadamente $[-3, 3](m/s^2)$ - figura 10. Este modelo aplica-se aos três acelerómetros com a pequena diferença de que no caso do sensor vertical, o valor de referência para a amplificação não é a 2,5 V mas sim a 1,5 V, para que o efeito da gravidade seja eliminado e portanto haja um melhor aproveitamento da excursão. Para que esta compensação por *hardware* faça sentido, o algoritmo de integração terá de desfazer está compensação por *software* antes de proceder à integração. Estes ganhos foram pensados por forma a maximizar a gama de utilização do IMU. Este foi pensado para um carro que se desloca dentro de uma cidade, com variações de dinâmica adequadas às condições normais de circulação.

Este circuito deverá ser implementado três vezes (à excepção do sensor de temperatura) e cada um deles será disposto de forma ortogonal relativamente aos outros, tal como se pode ver na figura 9.

Depois de acondicionados, os sinais são transferidos para o andar de sincronização (*sample and hold* SMP04), e seguidamente para o micro-controlador. Depois de convertidas, as medidas são enviadas via RS-232 (MAX232) para um computador responsável pelo processamento do algoritmo de integração - figura 11.

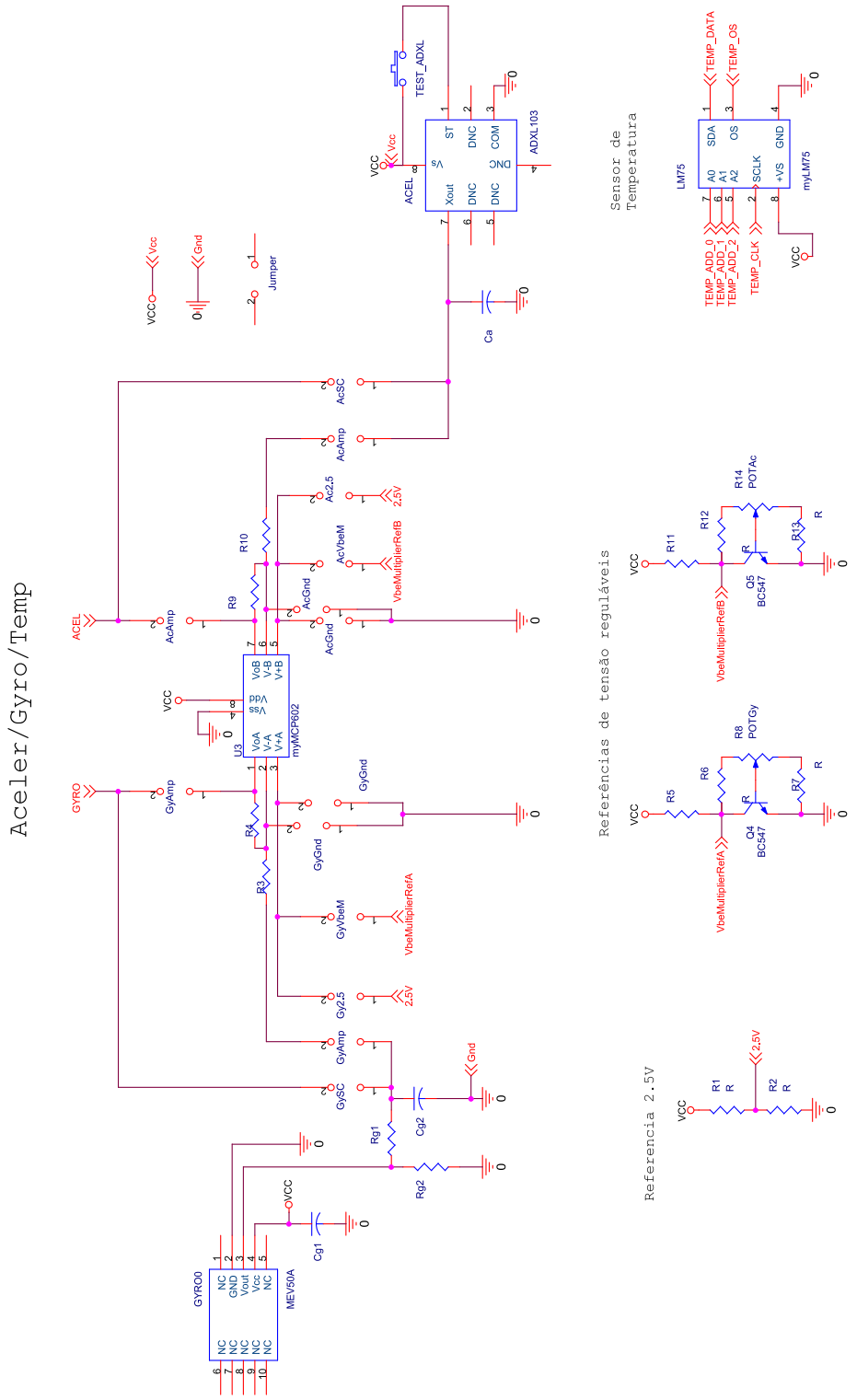

Figura 8: Circuito de acondicionamento dos sinais vindos do giroscópio, acelerómetro e sensor de temperatura

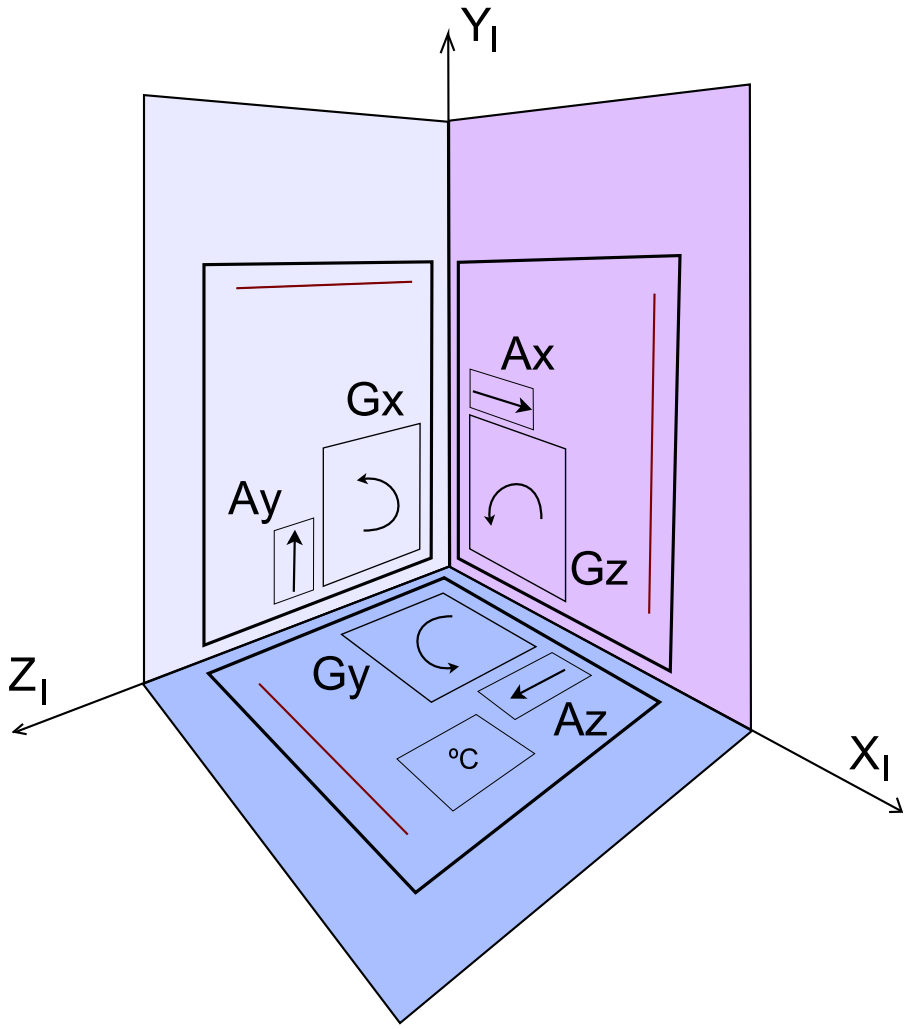

Figura 9: Disposição espacial dos sensores. (Ax, Ay, Az) e (Gx, Gy, Gz) representam vectores dos eixos de sensibilidade dos acelerómetros e os giroscópios, respectivamente, orientados em três direcções ortogonais $(\mathbf{X}_l, \mathbf{Y}_l, \mathbf{Z}_l)$.

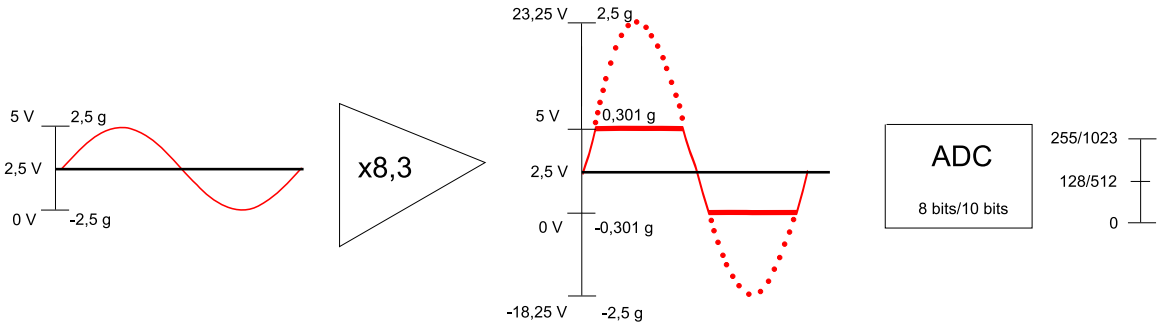

Figura 10: Andar de amplificação dos acelerómetros horizontais.

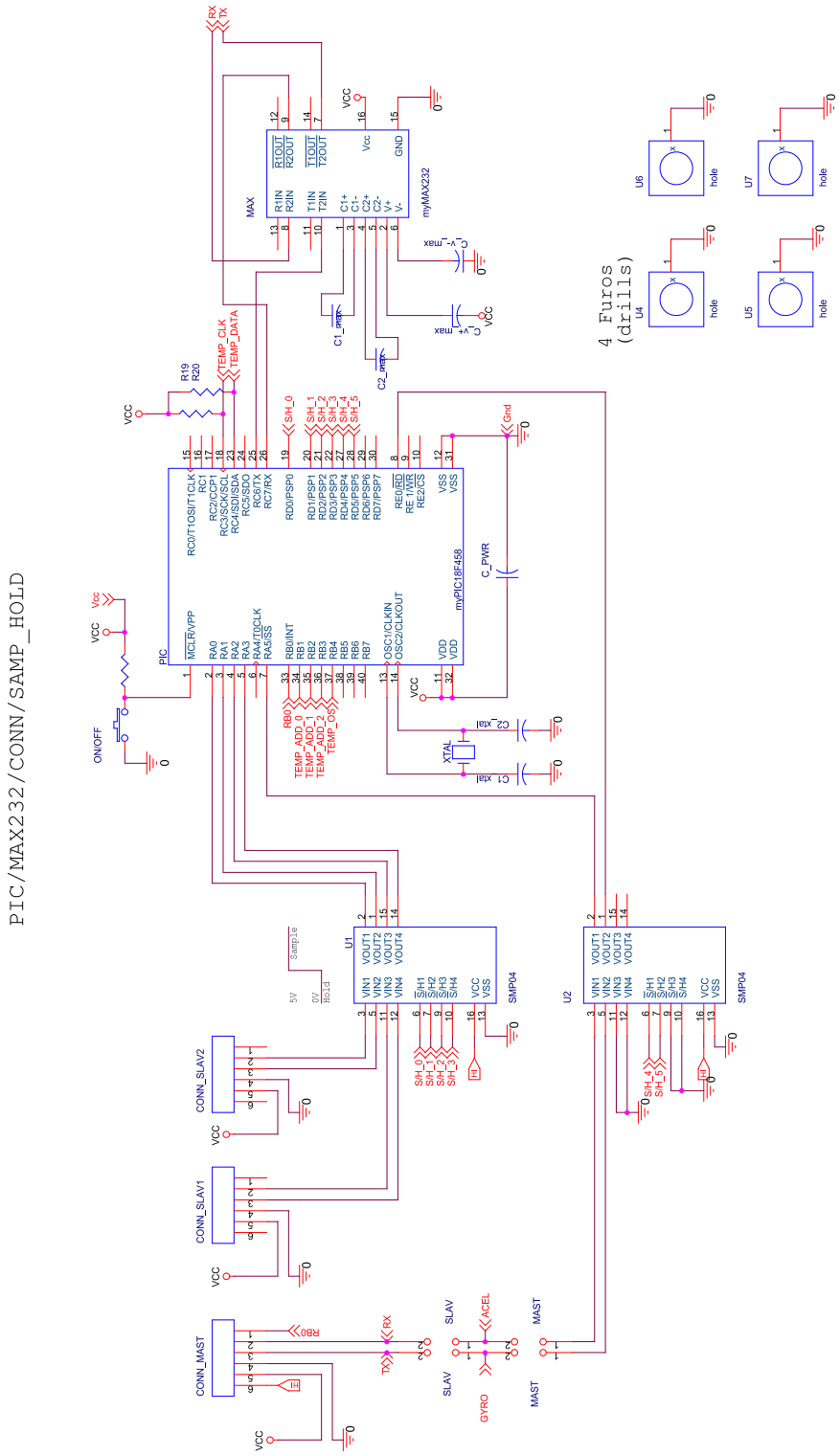

Figura 11: Circuito de recepção dos sinais após o acondicionamento e de transmissão de dados via RS-232

3.3 Projecto do PCB

A organização dos componentes foi feita no sentido de aproximar os sensores inerciais o mais possível, para minimizar as fontes de erro associadas ao afastamento dos eixos de sensibilidade dos sensores. O ideal seria colocar os seis sensores coincidentes num ponto com os eixos de sensibilidade dispostos ortogonalmente. Colocando o acelerómetro e o giroscópio o mais próximos de um dos cantos da PCB (figura 13) torna possível montar as três PCBs com base na configuração da figura 9.

A PCB foi concebida de forma a se poder, posteriormente, editar algumas ligações com *jumpers,* permitindo, assim, personalizar a PCB à função respectiva. Desta maneira implementaram-se três PCBs diferentes pelo preço de uma.

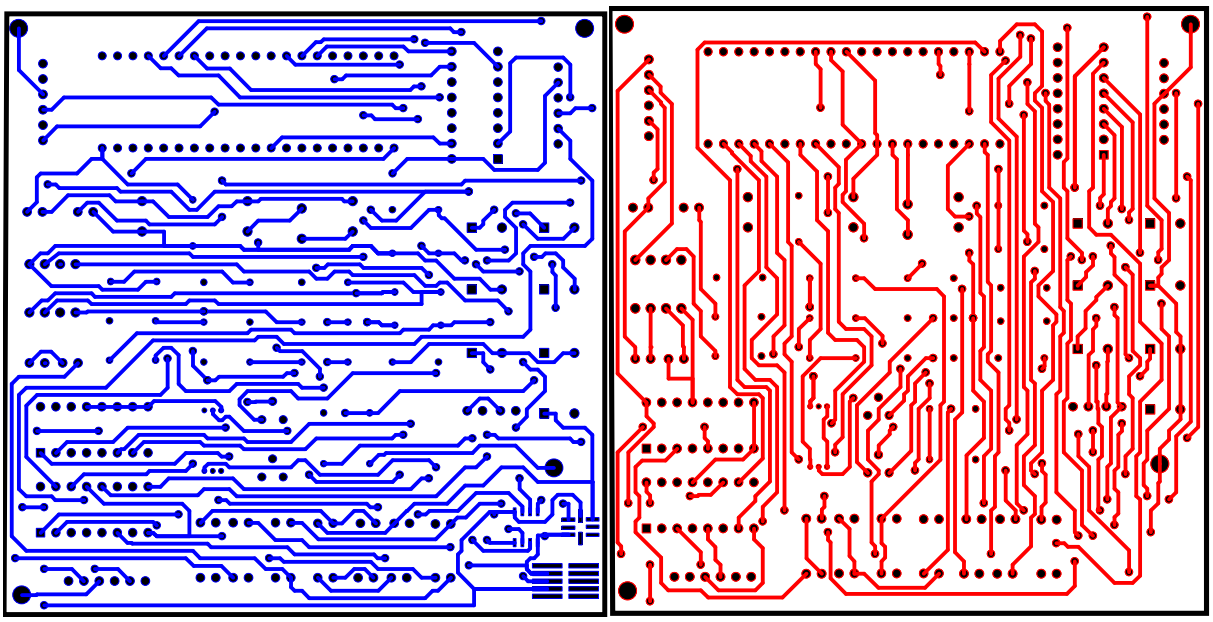

Figura 12: No lado esquerdo, a azul: pistas da PCB de topo;
No lado esquerdo, a vermelho: pistas da PCB por baixo.

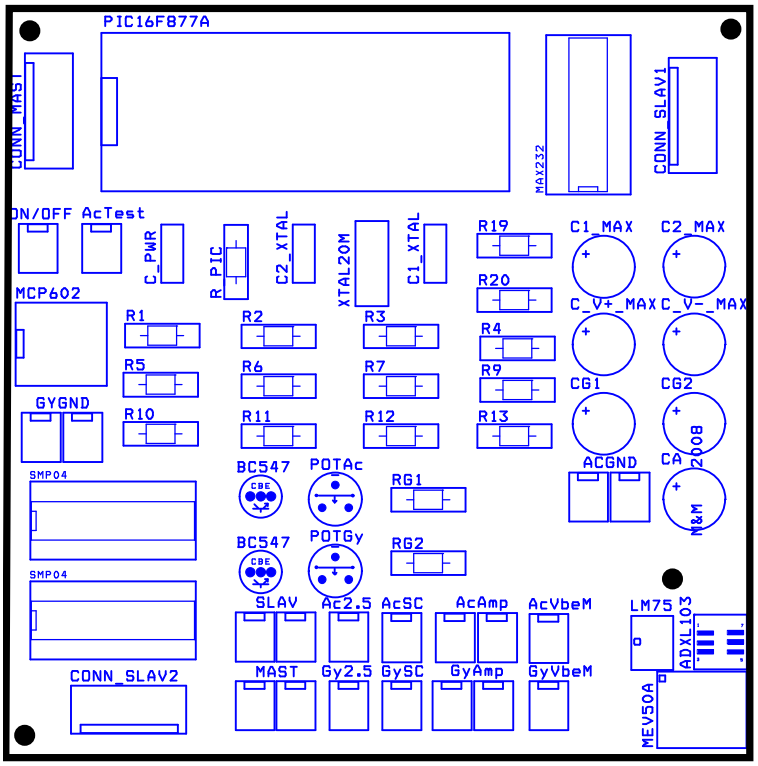

Figura 13: Disposição dos componentes na PCB

3.4 Produto final

O resultado final é uma estrutura móvel com base em madeira que suporta toda a electrónica envolvida para a concepção do IMU - figura 14.

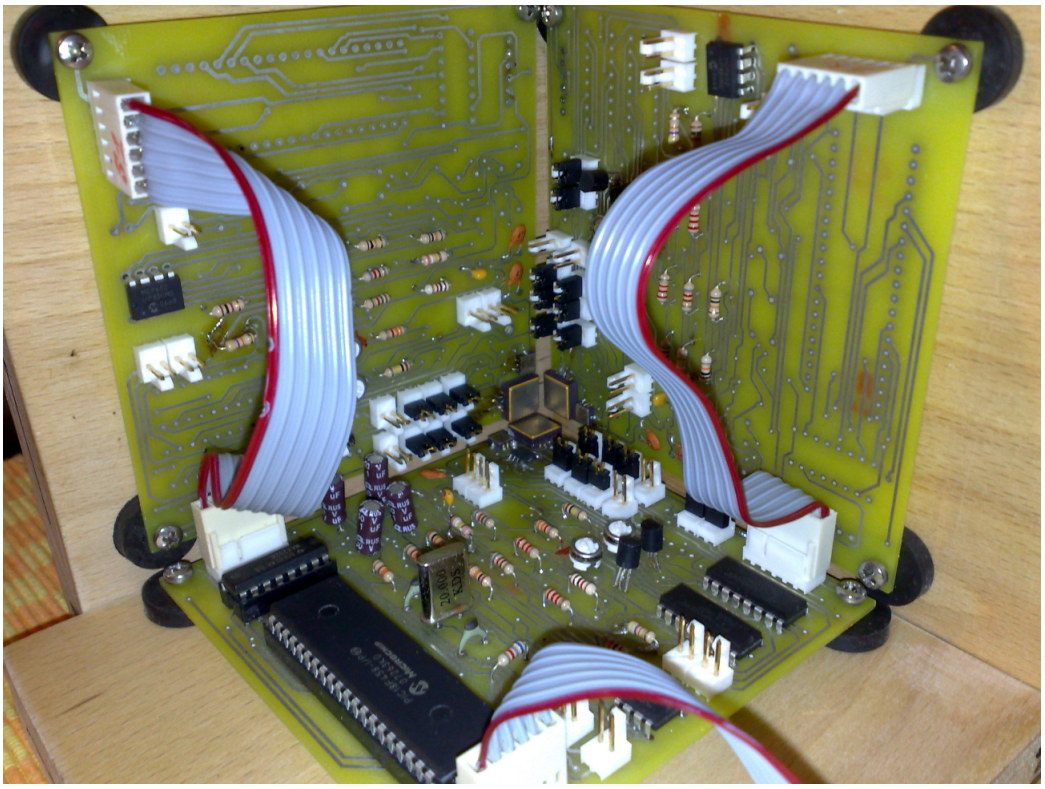

Figura 14: Fotografia do IMU.

3.5 Software do PIC - fluxograma

As funções principais do microprocessador são: fazer a recolha dos dados provenientes dos sensores inerciais através da ADC (6 canais) que lhe está incorporada, e das medidas de temperatura através do protocolo I2C. Depois de captados, os valores dos sensores, o microprocessador deve agrupar as medidas numa trama predefinida e enviá-las pela porta série. Este procedimento é repetido a uma cadência de 100Hz. No início de cada turno as medidas inerciais dos seis canais são "memorizadas" por *sample & hold's*, permitindo à ADC converter os dados, do mesmo instante, de cada sensor, sequencialmente - figura 15.

Todo o procedimento funcionará com base num algorítmo temporal com a qualidade de se poder sincronizar a partir de um sinal externo, oriundo de um GPS (1PPS - 1 pulso por segundo). Este sinal de sincronismo é utilizado para sincronizar todos os sensores utilizados no sistema de navegação para que, ao integrar os dados de todos, em conjunto, estes se reportem aos mesmos instantes de tempo efectivos. No micro-controlador, o sinal de sincronismo produz uma interrupção e a rotina que atende essa interrupção efectua o acerto do relógio interno no PIC com o valor absoluto do tempo que é recebido via porta

série RS232.

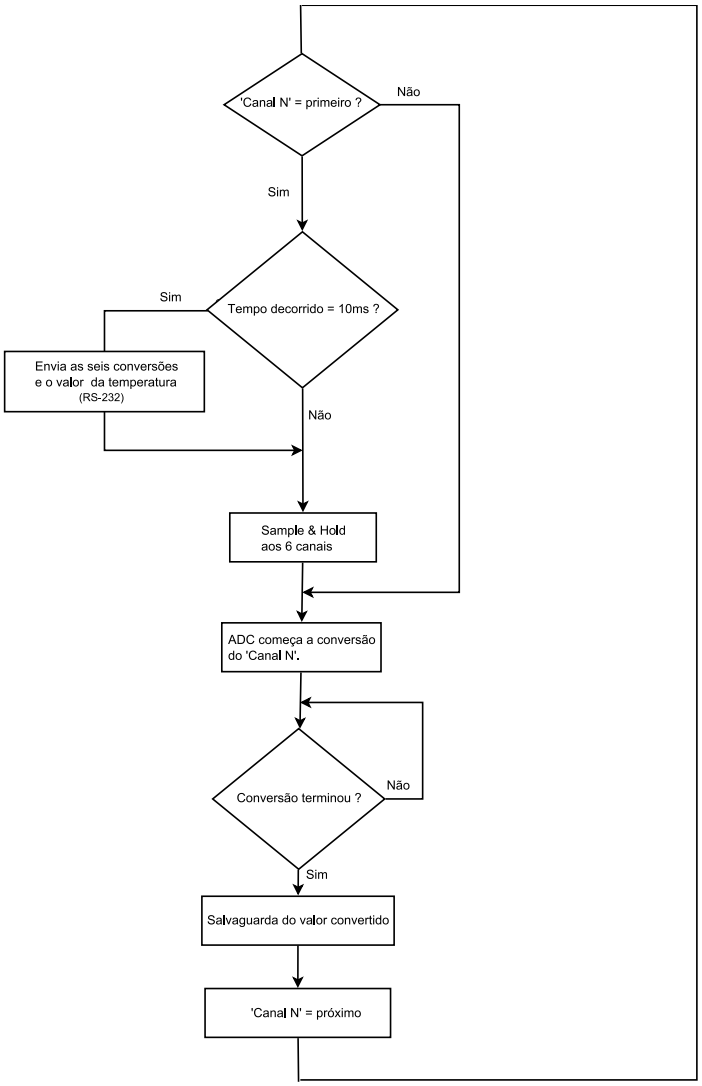

Figura 15: Fluxograma do algoritmo do microprocessador.

4 Algoritmo de processamento dos dados

O algoritmo adoptado para o processamento da informação em sistemas *strapdown* tem essencialmente quatro fases como se pode ver na figura 16. Na primeira determina-se a orientação do dispositivo fazendo uso dos sinais captados dos giroscópios, na segunda fase usa-se a orientação previamente calculada para projectar as medidas vindas dos acelerómetros no referencial global. De seguida faz-se a compensação das medidas para eliminar a influência da gravidade e finalmente aplica-se a dupla integração por forma a obter a posição relativamente às condições iniciais estabelecidas.

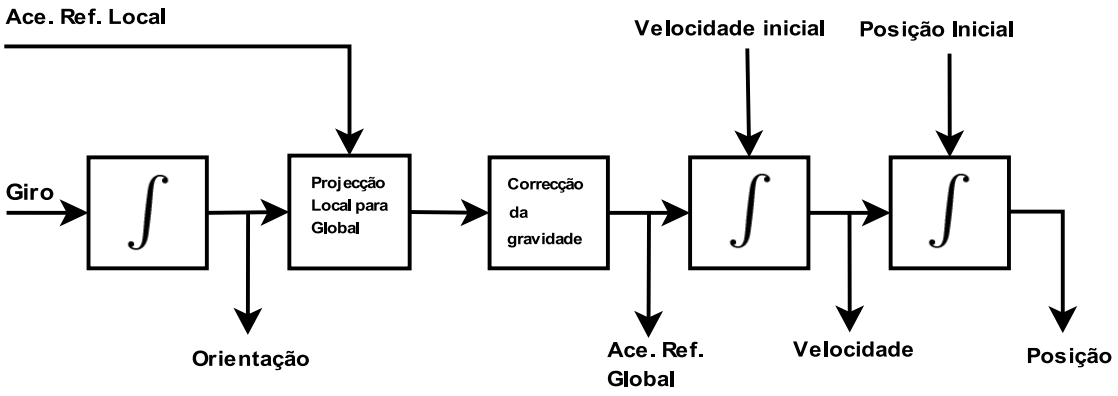

Figura 16: Algoritmo aplicado ao INS *strapdown* (adaptado de [2])

Nesta secção a letra 'l' será usada em índice de vectores que se refiram ao referencial local. O mesmo se aplica à letra 'g', mas neste caso os vectores referir-se-ão ao referencial global, tal como indica a figura 17.

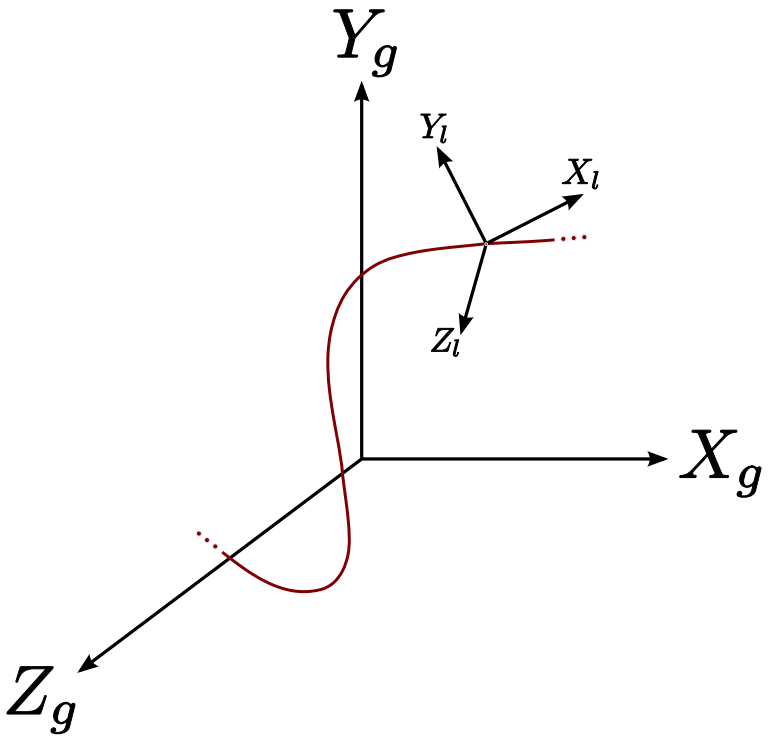

Figura 17: Referencial local e referencial global.
$(\mathbf{X}_l, \mathbf{Y}_l, \mathbf{Z}_l)$: referencial local
$(\mathbf{X_g}, \mathbf{Y_g}, \mathbf{Z_g})$:referencial global

4.1 Cálculo da Orientação

4.1.1 Teoria

Para representar a orientação do IMU serão usados os cossenos directores, isto é, a atitude do referencial local será descrita pela matriz de projecção C, 3×3, cujas colunas correspondem aos versores directores dos eixos do referencial local, X_l, Y_l, Z_l. Desta forma, para projectar um dado vector v_l,definido relativamente ao referencial local, no referencial global basta calcular

$$\mathbf{v}_g = \mathbf{C} \cdot \mathbf{v}_l \tag{7}$$

Para que a atitude seja bem determinada a matriz C tem de ser actualizada com a mesma frequência que são amostradas as velocidades angulares provenientes dos giroscópios $\omega_l(t) = (\omega_{lx}(t), \omega_{ly}(t), \omega_{lz}(t))^T$. Sendo $C(t)$ a matriz orientação no instante t pode-

se exprimir a variação de C pela definição de derivada

$$\dot{\mathbf{C}}(t) = \lim_{\delta t \to 0} \frac{\mathbf{C}(t+\delta t) - \mathbf{C}(t)}{\delta t} \tag{8}$$

em que δt corresponde ao período de amostragem. Depois, é possível exprimir $C(t+\delta t)$ como sendo a multiplicação da matriz de projecção da iteração anterior, $C(t)$, com a matriz rotação $A(t)$

$$\mathbf{C}(t+\delta t) = \mathbf{C}(t)\,\mathbf{A}(t) \tag{9}$$

em que

$$\mathbf{A}(t) = \mathbf{I} + \delta\mathbf{\Psi} \tag{10}$$

e

$$\delta\mathbf{\Psi} = \begin{pmatrix} 0 & -\delta\psi & \delta\theta \\ \delta\psi & 0 & -\delta\phi \\ -\delta\theta & \delta\phi & 0 \end{pmatrix} \tag{11}$$

É de notar que $A(t) = I + \delta\Psi$ só se verifica para ângulos de rotação, $\delta\phi, \delta\theta$ e $\delta\psi$, muito pequenos.[1]

Fazendo as substituições devidas:

[1]Um dado vector v quando multiplicado por uma matriz rotação altera a sua direcção mantendo a norma. As três matrizes responsáveis por rodar um vector no espaço 3-D são as seguintes:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix}$$

$$\mathbf{R}_y = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix}$$

$$\mathbf{R}_z = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Uma rotação à custa de ângulos positivos significa rodar em torno do eixo das abcissas aplicando a regra-da-mão-direita com o polegar a apontar no sentido positivo do eixo. Desta forma, qualquer rotação no espaço pode ser representada pela multiplicação das três matrizes previamente descritas

$$\mathbf{v}_R = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z \cdot \mathbf{v}$$

sendo v_R o vector v após a rotação. No entanto, a ordem pela qual as matrizes são multiplicadas deve ser respeitada, isto é, rotações no espaço tridimensional não são comutativas. Por outro lado, para ângulos de rotação muito pequenos o resultado de $sin(\phi), sin(\theta)$ e $sin(\psi)$ aproxima-se de ϕ, θ e ψ respectivamente e os cossenos destes ângulos tenderão para 1. Feitas as substituições, as multiplicação entre as matrizes

$$\dot{\mathbf{C}} = \lim_{\delta t \to 0} \frac{\mathbf{C}(t+\delta t) - \mathbf{C}(t)}{\delta t} \tag{12}$$

$$= \lim_{\delta t \to 0} \frac{\mathbf{C}(t)\,\mathbf{A}(t) - \mathbf{C}(t)}{\delta t} \tag{13}$$

$$= \lim_{\delta t \to 0} \frac{\mathbf{C}(t)\,(\mathbf{I} + \delta\boldsymbol{\Psi}) - \mathbf{C}(t)}{\delta t} \tag{14}$$

$$= \mathbf{C}(t) \cdot \lim_{\delta t \to 0} \frac{\delta\boldsymbol{\Psi}}{\delta t} \tag{15}$$

Como a taxa a que são captadas as medidas é suficientemente elevada, a aproximação de pequenos ângulos pode ser tida em consideração

$$\lim_{\delta t \to 0} \frac{\delta\boldsymbol{\Psi}}{\delta t} = \boldsymbol{\Omega}(t) \tag{16}$$

onde

$$\boldsymbol{\Omega}(t) = \begin{pmatrix} 0 & -\omega_{lz}(t) & \omega_{ly}(t) \\ \omega_{lz}(t) & 0 & -\omega_{lx}(t) \\ -\omega_{ly}(t) & \omega_{lz}(t) & 0 \end{pmatrix} \tag{17}$$

Finalmente, substituindo, está-se perante a equação diferencial

$$\dot{\mathbf{C}}(t) = \mathbf{C}(t)\,\boldsymbol{\Omega}(t) \tag{18}$$

com a seguinte solução

$$\mathbf{C}(t) = \mathbf{C}(0) \cdot exp\left(\int_0^t \boldsymbol{\Omega}(t)\,dt\right) \tag{19}$$

em que $C(0)$ é a orientação inicial do dispositivo.

4.1.2 Prática

Na prática, as velocidades angulares são captadas a uma dada frequência de amostragem e por isso estes sinais não são contínuos, mas sim discretos. Visto isto, a solução aqui apresentada assume como constantes as velocidades angulares durante o período de amostragem

rotação passam a ser comutativas e de valor

$$\mathbf{R} = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z \approx \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{pmatrix}$$

δt, até que novas medidas sejam adquiridas. Para o período $[t, t+\delta t]$, a solução da equação (2.13) é a seguinte

$$\mathbf{C}(t+\delta t) = \mathbf{C}(t) \cdot exp\left(\int_t^{t+\delta t} \mathbf{\Omega}(t)\, dt\right) \tag{20}$$

onde

$$\int_t^{t+\delta t} \mathbf{\Omega}(t)\, dt = \mathbf{B} \tag{21}$$

com

$$\mathbf{B} = \begin{pmatrix} 0 & -\omega_{lz}\delta t & \omega_{ly}\delta t \\ \omega_{lz}\delta t & 0 & -\omega_{lx}\delta t \\ -\omega_{ly}\delta t & \omega_{lx}\delta t & 0 \end{pmatrix} \tag{22}$$

Aplicando a expansão em série de Taylor da função exponencial obtém-se

$$\begin{aligned}
\mathbf{C}(t+\delta t) &= \mathbf{C}(t)\left(\mathbf{I} + \mathbf{B} + \frac{\mathbf{B}^2}{2!} + \frac{\mathbf{B}^3}{3!} + \frac{\mathbf{B}^4}{4!} + \ldots\right) && (23)\\
&= \mathbf{C}(t)\left(\mathbf{I} + \mathbf{B} + \frac{\mathbf{B}^2}{2!} - \frac{\sigma^2\mathbf{B}}{3!} - \frac{\sigma^2\mathbf{B}^2}{4!} + \ldots\right) && (24)\\
&= \mathbf{C}(t)\left(\mathbf{I} + \left(1 - \frac{\sigma^2}{3!} + \frac{\sigma^4}{5!}\ldots\right)\mathbf{B} + \left(\frac{1}{2!} - \frac{\sigma^2}{4!} + \frac{\sigma^4}{6!}\ldots\right)\mathbf{B}^2\right) && (25)\\
&= \mathbf{C}(t)\left(\mathbf{I} + \frac{sin(\sigma)}{\sigma}\mathbf{B} + \frac{1 - cos(\sigma)}{\sigma^2}\mathbf{B}^2\right) && (26)
\end{aligned}$$

em que $\sigma = |\omega_l \delta t|$ e $\omega_l = (\omega_{lx}, \omega_{ly}, \omega_{lz})^T$. A equação (2.20) será, então, usada para actualizar a matriz projecção $\mathbf{C}$ todas as iterações.

4.2 Cálculo da Posição

4.2.1 Teoria

Para que as acelerações provenientes dos acelerómetros $\mathbf{a}_l(t) = (a_{lx}(t), a_{ly}(t), a_{lz}(t))^T$ sejam correctamente integradas, estas devem ser projectadas no referencial global da seguinte maneira:

$$\mathbf{a}_g(t) = \mathbf{C}(t)\,\mathbf{a}_l(t) \tag{27}$$

Depois, com as medidas já referenciadas aos eixos globais, aplica-se a dupla integração e faz-se a compensação da aceleração da gravidade

$$\mathbf{v}_g(t) = \mathbf{v}_g(0) + \int_0^t (\mathbf{a_g}(t) - \mathbf{g}_g)\, dt \tag{28}$$

$$\mathbf{p}_g(t) = \mathbf{p}_g(0) + \int_0^t \mathbf{v}_g(t)\, dt \tag{29}$$

onde $\mathbf{v}_g(0)$ e $\mathbf{p}_g(0)$ são a velocidade e a posição iniciais (condições iniciais) e $\mathbf{g}_g$ é a aceleração devido à gravidade.

4.2.2 Prática

Assumindo, então, que os valores das acelerações são constantes durante o período δt, a integração é feita da seguinte maneira:

$$\mathbf{v}_g(t+\delta t) = \mathbf{v}_g(t) + \delta t \cdot (\mathbf{a}_g(t+\delta t) - \mathbf{g}_g) \tag{30}$$

$$\mathbf{p}_g(t+\delta t) = \mathbf{p}_g(t) + \delta t \cdot \mathbf{v}_g(t+\delta t) \tag{31}$$

	Média		
Eixo de sensibilidade	x	y	z
Giroscópios $(°/s)$	-6.225×10^{-3}	7.948×10^{-4}	4.904×10^{-4}
Acelerómetros (m/s^2)	4.930×10^{-4}	6.790×10^{-4}	9.069×10^{-4}

Tabela 2: Média relativa a medidas do giroscópio e acelerómetro quando em repouso.

	Desv. Padrão		
Eixo de sensibilidade	x	y	z
Giroscópios $(°/s)$	1.424×10^{-3}	1.900×10^{-5}	2.438×10^{-5}
Acelerómetros (m/s^2)	3.169×10^{-6}	2.642×10^{-5}	1.006×10^{-4}

Tabela 3: Desvio padrão relativo a medidas do giroscópio e acelerómetro quando em repouso.

5 Resultados

Aqui mostram-se alguns resultados após a implementação do IMU: efectuou-se uma análise estatística com os sensores em repouso e outra quando sujeitos a variações de temperatura. Nas seguintes tabelas (2 e 3) encontram-se os resultados destas medições.

É, também, possível verificar a importância de um sensor de temperatura de maneira a poder fazer-se uma compensação dos desvios acumulados por variações da temperatura.

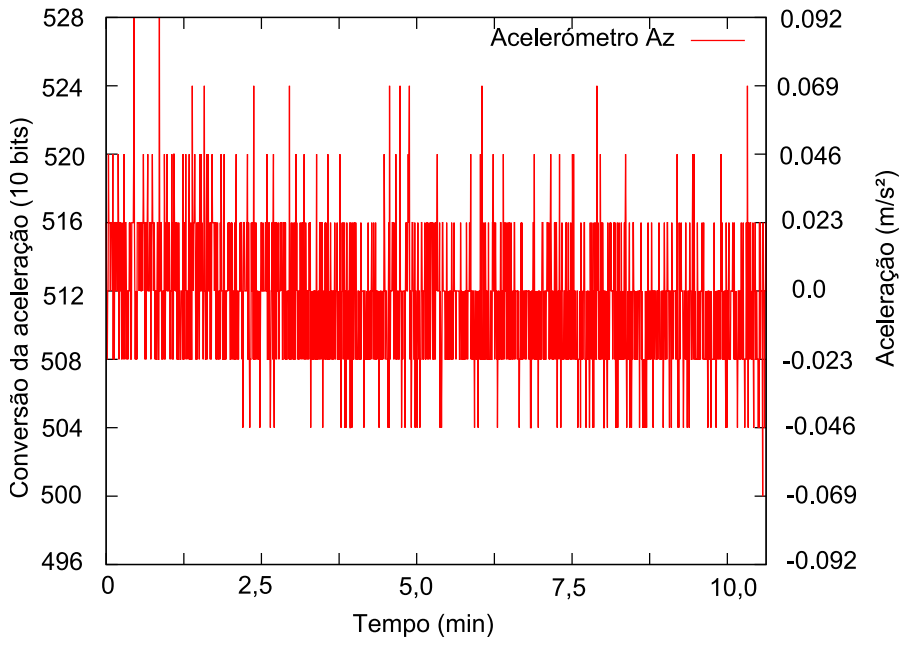

Figura 18: Valor medido do acelerómeto Az (horizontal) com o IMU em repouso, à temperatura ambiente (20^{o}C).

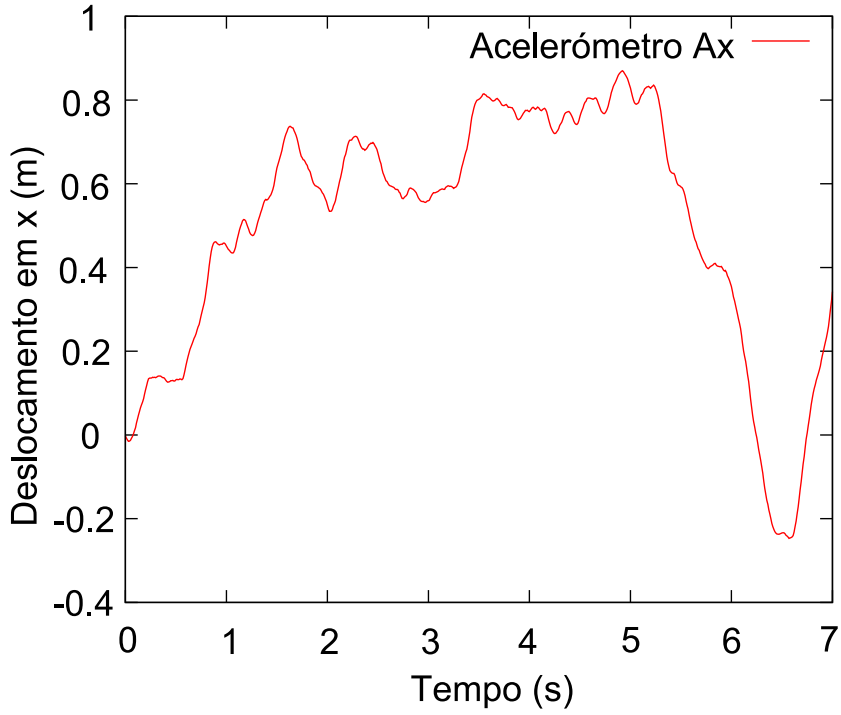

Figura 19: Random walk que advém da dupla integração das acelerações que provêm do acelerómetro Ax.

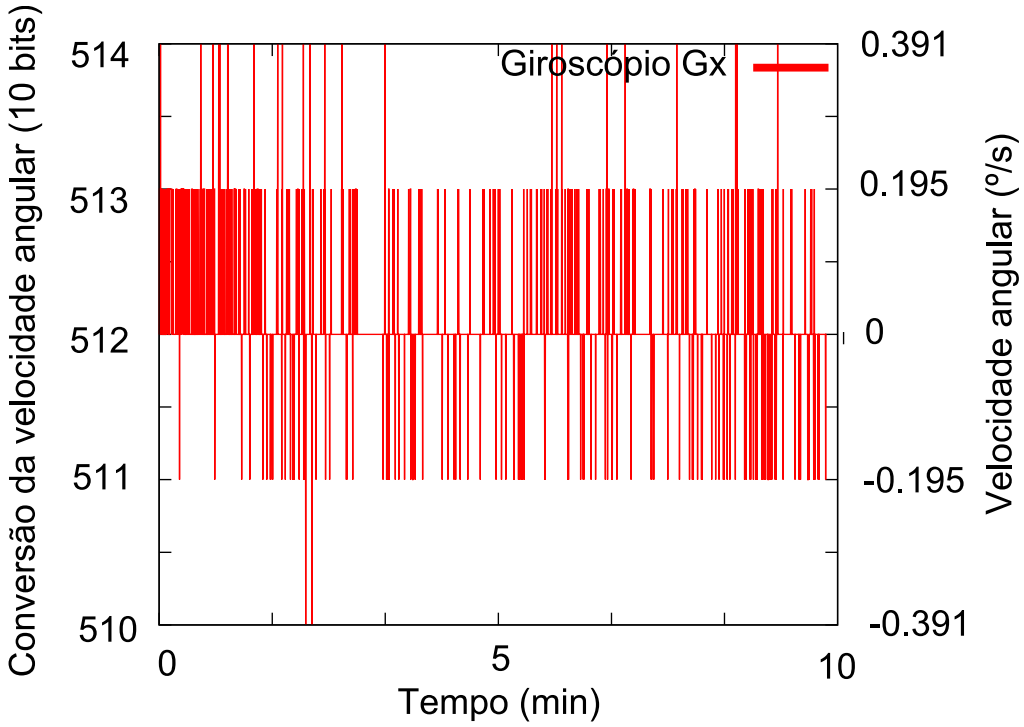

Figura 20: Medidas obtidas do giroscópio Gx quando em repouso, à temparatura constante de 20ºC.

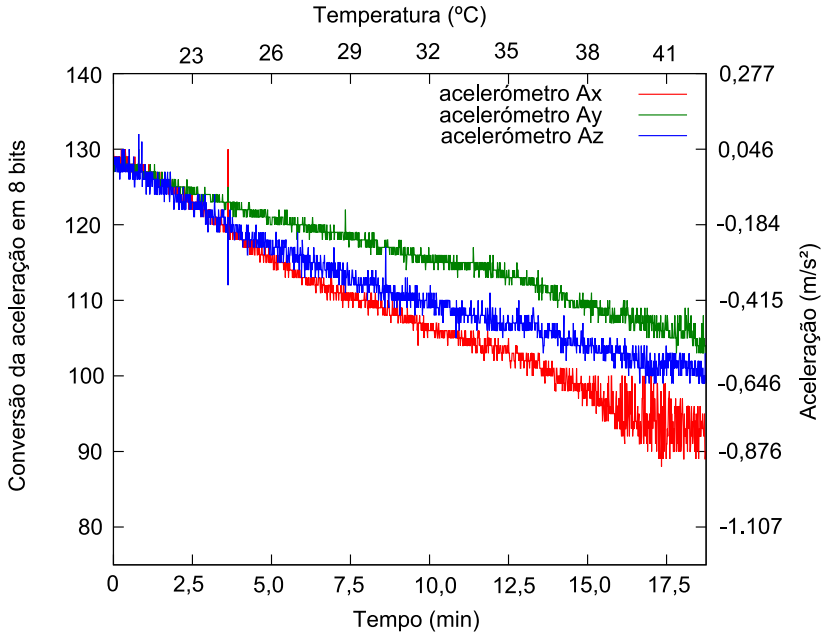

Figura 21: Influência gerada por variações da temperatura nos acelerómetros.

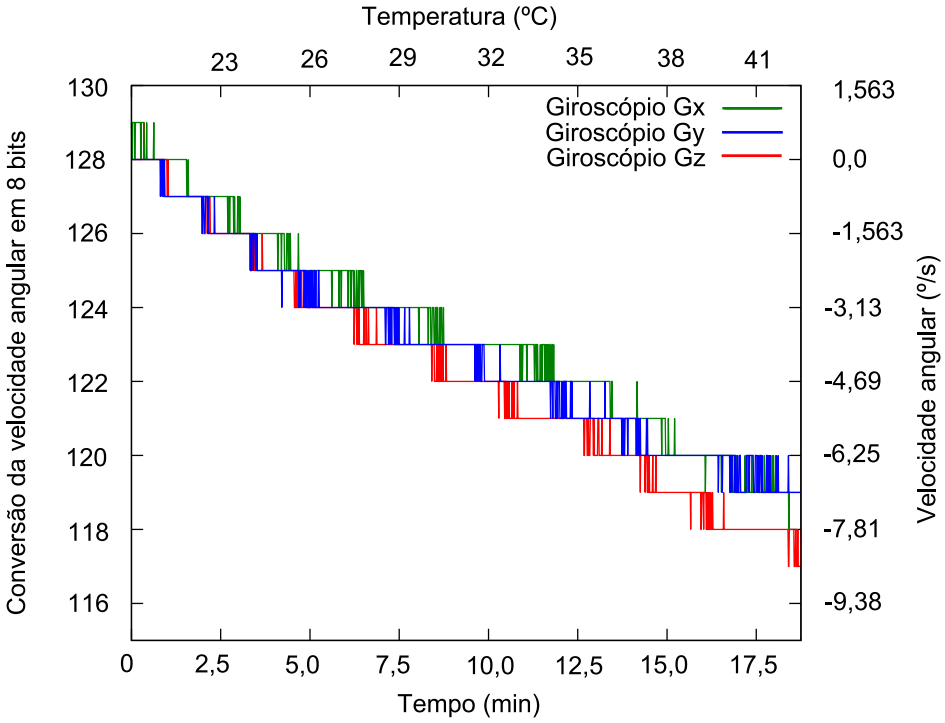

Figura 22: Influência gerada por variações da temperatura nos giroscópios.

Para fazer a análise da influência da temperatura nos sinais dos sensores, introduziu-

se o IMU dentro de um volume com um tamanho suficientemente grande para o cobrir. Posteriormente aqueceu-se o ar dentro do volume com um ferro de soldar, conseguindo-se uma variação uniforme da temperatura do ar em torno dos sensores, sem variações bruscas.

6 Conclusões

Hoje em dia, a tecnologia MEMS ainda não permite criar sistemas inerciais que mantenham um erro de posição inferior a 1 metro, ao fim de 1 minuto, quando em repouso. Foi visto que este erro gerado, mesmo quando o IMU está parado, deve-se ao desvio que a integração do ruído dos sensores inerciais proporciona. Durante este projecto registou-se, também, que outra forte componente do erro se deve à imperfeita ortogonalidade dos sensores entre si, consequência de toda a electrónica envolvida ter sido montada manualmente. Por outro lado, apesar de um IMU não ser capaz de determinar a posição garantindo um erro constante, este pode ser uma boa solução quando complementado com outros sistemas de posicionamento, tais como o GPS. Deste modo, o desvio gerado pelo IMU corrige-se periodicamente à custa das medições absolutas do GPS e as zonas mortas compreendidas entre sucessivas actualizações do GPS passam a ser conhecidas com a contribuição do IMU. Esta cooperação é extremamente interessante, pois permite complementar um GPS, reduzindo significativamente o erro, tornando-o comparável a GPSs de maior qualidade e custo. Este IMU, pode ser uma mais valia em experiências futuras que envolvam outros dispositivos de navegação e outros algoritmos.

7 Apêndice A

Este programa é responsável pelo recepção, armazenamento e integração das medidas extraidas dos sensores.

```
# # # # #
# NOTES #
# # # # #
#                                   ____________
#                                   -_PROTOCOL_-
#  ___________ ____________________________ ___________________ ______
# |    header |                  measures  |         time      |chksum|
# |EA|BB|00|0B|-Ax|Wbz|-Ay|Wbx|-Az|Wby|Temp|MsByte|    |LsByte|CHKSUM|
#
#
#
#
#!C:/Programas/Perl/bin/perl.exe
use strict;
use Win32::SerialPort;
use Math::Matrix;
use Time::HiRes qw(gettimeofday tv_interval);
use POSIX qw(ceil floor);
# constants
use constant PI             => 4 * atan2(1, 1);
use constant X_OFFSET_LIN   => 128;
 use constant Y_OFFSET_LIN  => 128;
use constant Z_OFFSET_LIN   => 128;
use constant X_OFFSET_ANG   => 128;
use constant Y_OFFSET_ANG   => 129;
use constant Z_OFFSET_ANG   => 128;
# variables
my $out;
my $port;
my @matched;
my $var;
my $checksum;
my $chk_aux;
my $a=chr(0xea);
my $b=chr(0xbb);
my $c=chr(0x00);
my $d=chr(0x0b);
my $t_start;
my $t0;
my $t1;
my @filter;
my $count_filter = 0;
my $B                 = new Math::Matrix([0,0,0], [0,0,0], [0,0,0]);
my $C                 = new Math::Matrix([1,0,0], [0,1,0], [0,0,1]);
my $ang_velocity      = new Math::Matrix([0,0,0]);
my $lin_acceleration  = new Math::Matrix([0,0,0]);
my $lin_accel_proj    = new Math::Matrix([0,0,0]);
my $lin_vel_proj      = new Math::Matrix([0,0,0]);
my $lin_pos_proj      = new Math::Matrix([0,0,0]);
my $temperature;
my $time;
# My way variables
my $pos_1    = new Math::Matrix([0,0,0]);
my $pos      = new Math::Matrix([0,0,0]);
my $vel_1    = new Math::Matrix([0,0,0]);
my $vel      = new Math::Matrix([0,0,0]);
# Raw Measures
my $raw_lin_acc = new Math::Matrix([0,0,0]);
my $raw_ang_vel = new Math::Matrix([0,0,0]);
my $raw_temperature;
my $raw_time;

# Calibration Variables my @offset_lin_sum = (0,0,0);
my @offset_ang_sum = (0,0,0);
```

```
my @offset_lin_calibrated = (512,512,512);
my @offset_ang_calibrated = (512,512,512);
#########################################################################
# MAIN
# START TIME COUNTING
$t_start = time;
# COMMUNICATION (rs232)
$port = new Win32::SerialPort ("COM3");
$port->baudrate(115200);
$port->databits(8);
$port->stopbits(1);
$port->parity("none");
while (1){
   $out = $port->input;
   if($out ne ''){
     # INPUT MEASURES # # # # # # # # # # # # # # # #
 $var = $var.$out;
if($var=~/($a$b$c$d)(.{1})(.{1})(.{1})(.{1})(.{1})(.{1})
                     (.{1})(.{1})(.{1})(.{1})(.{1})(.{1})
                     (.{1})(.{1})(.{1})(.{1})(.{1})(.{1})(.*)/o) {
      $checksum=(0xea+0xbb+0x0b+ord($2)+ord($3)+ord($4)+
                 ord($5)+ord($6)+ord($7)+ord($8)+ord($9)+
                 ord($10)+ord($11)+ord($12)+ord($13)+ord($14)+
                 ord($15)+ord($16)+ord($17)+ord($18))&(0xFF);
         $chk_aux = $19;
#        CHECKSUM TRUE ?
         if($checksum == ord($chk_aux)){
#           print "CHKSUM TRUE!";
            $var = $20;
         @matched = ($1,   $2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,    $14,   $15,$16,$17,$18,     $19);
#           print sprintf("%x", ord($matched[11]) << 8);
            # TREAT MEASURES # # # # # # # # # # # # # # #
            ($raw_ang_vel, $raw_lin_acc, $raw_temperature, $raw_time) = GetRawMeasures(\@matched);
            ($ang_velocity, $lin_acceleration, $temperature, $time) = GetMeasures(\@matched);
#           $lin_acceleration->[0][1] = 1 - $lin_acceleration->[0][1]*0.7/0.3;
#           $lin_acceleration->print("");
#           print $lin_acceleration->[0][0],"\t",$lin_acceleration->[0][1],"\t",$lin_acceleration->[0][2],"\n";
#           print $lin_acceleration->[0][0],"\t",$lin_acceleration->[0][1],"\t",$lin_acceleration->[0][2],"\n";
#           $raw_lin_acc->print("");
#           $raw_ang_vel->print("");
#           print sprintf("TIME: %i",$time);
#           $ang_velocity->print("Ang_vel:");
#           $lin_acceleration->print("lin_ace:");
#           print $temperature;
#           print $time;
#           print values
=pod
            foreach (@matched){
               print iso2hex($_);
            }
            print "\n";
=cut
#           PROCESS INPUT # # # # # # # # # # # # # # # # #
=pod
            $ang_velocity->[0][0] = 0;
            $ang_velocity->[0][1] = 0;
            $ang_velocity->[0][2] = 0;
#=cut
#           $ang_velocity->[0][1] += 0.06;
=pod              $lin_acceleration->[0][0] = 0;
            $lin_acceleration->[0][1] = 0;
            $lin_acceleration->[0][2] = 0;
=cut
#           $lin_acceleration->[0][0] += 1;
            if( (time - $t_start) > $ARGV[0]){
#              print " DONE !";  #AFTER warm up time

               # Calculate Mtx B
               $B = CalcMtxB($ang_velocity);

               # Calculate Mtx C
```

```
                $C = CalcMtxC($ang_velocity, $B, $C);
                # $lin_acceleration->print("acc:\n");

                # Project Acceleration
                $lin_accel_proj = ProjAccel($lin_acceleration, $C);
                # $lin_accel_proj->print("l a p:");
=pod
            ######################################################################
            # # # # #
            # MY WAY
                # Integrates Acceleration
                $vel_1 = MyIntAcc($lin_accel_proj, $vel);
                # Integrates Velocity
                $pos_1 = MyIntVel($lin_accel_proj, $vel, $pos);

                $vel = $vel_1;
                $pos = $pos_1;
            # # # # # # # # # # #
=cut
            ######################################################################
            # # # # # # #
            # ARTICLE WAY: Euler method
                # Integrates Acceleration
                $lin_vel_proj = IntegAcceleration($lin_vel_proj, $lin_accel_proj);
                # $lin_vel_proj->print("vel proj");
                # Integrates Velocity
                $lin_pos_proj=IntegVelocity($lin_vel_proj,$lin_pos_proj);
            ######################################################################
            }
            else{
                $offset_ang_sum[0] += $raw_ang_vel->[0][0]/10.0;
                $offset_ang_sum[1] += $raw_ang_vel->[0][1]/10.0;
                $offset_ang_sum[2] += $raw_ang_vel->[0][2]/10.0;
                $offset_lin_sum[0] += $raw_lin_acc->[0][0]/10.0;
                $offset_lin_sum[1] += $raw_lin_acc->[0][1]/10.0;
                $offset_lin_sum[2] += $raw_lin_acc->[0][2]/10.0;
                if (++$count_filter > 9){
                # print "@filter\n";
                # printf("%.1f\t%.1f\t%.1f\t%.1f\t%.1f\t%.1f\n",
$offset_ang_sum[0],
$offset_ang_sum[1],
$offset_ang_sum[2],
$offset_lin_sum[0],
$offset_lin_sum[1],
$offset_lin_sum[2]);
                @offset_ang_calibrated = ($offset_ang_sum[0], $offset_ang_sum[1], $offset_ang_sum[2]);
@offset_lin_calibrated = ($offset_lin_sum[0], $offset_lin_sum[1], $offset_lin_sum[2]);
                @offset_ang_sum = (0, 0, 0);
                @offset_lin_sum = (0, 0, 0);
                $count_filter = 0;                }
            }
            # # # # # # # # # # #
            # $pos->print("My pos: $time ");
            # $lin_vel_proj->print("vel: $time ");
            # $lin_vel_proj->print("");
            $lin_pos_proj->print("");
            # $lin_accel_proj->print("");
            # $B->print ("B:\n");
            # $C->print("\n");
            # $raw_lin_acc->print("");
 =pod
            (undef, $t1) = gettimeofday;
            print "elapsed: ", ($t1 - $t0), "\n";
            $t0 = $t1;
=cut
            if ((time - $t_start) >= $ARGV[1]){
                exit;
            }
        }
        else{
            print "CHECKSUM ERROR!";
```

```
            # print sprintf ("%2x\n" ,$checksum);
             print sprintf ("%2x\n" ,$checksum) ," ", ord($chk_aux);
            # print sprintf ("%d\n" ,$chk_aux), "\n";
          }
       }
    }
}
#print $var;
#########################################################################
# FUNCTIONS
###################################
## Integrates Acceleration (MY WAY)
sub MyIntAcc {
    my $l_a    = $_[0];
    my $l_v    = $_[1];
    my $grav       = new Math::Matrix([0.0, -1.0, 0.0]);
    return (($l_a->subtract($grav))->multiply_scalar(0.010))->add($l_v));
}
###################################
## Integrates Acceleration (MY WAY)
sub MyIntVel {
    my $l_a       = $_[0];
    my $l_v       = $_[1];
    my $l_p       = $_[2];
    my $grav       = new Math::Matrix([0.0, -1.0, 0.0]);
    return ((($l_a->subtract($grav))->multiply_scalar(0.010*0.010*0.5))->add($l_v->multiply_scalar(0.010)))->add($l_p));
}
###################################
## Integrates Acceleration
sub IntegAcceleration {
    my $l_v_p    = $_[0];
    my $l_a_p    = $_[1];
    my $grav       = new Math::Matrix([0.0, -9.8, 0.0]);
#    $l_a_p->print("lin ace prj:  ");
#($l_a_p->subtract($grav))->print("lin ace proj: ");
#    $l_a_p->print("acc:");
    # Vg(t+£t) = Vg(t) + £t*(Ag(t+£t) - Gg)
    return ( (($l_a_p->add($grav) )->multiply_scalar(0.010))->add($l_v_p) );
}
###################################
## Integrates Velocity
sub IntegVelocity {
    my $l_v_p    = $_[0];
    my $l_p_p    = $_[1];
    return ( ($l_v_p->multiply_scalar(0.010))->add($l_p_p) );
}
###################################
## Get RAW Measures from @matched | No units convertions are done!
sub GetRawMeasures {
    my @m = @{$_[0]};
    my $a_v = new Math::Matrix([(ord($m[7]) << 8 ) + ord($m[8]), (ord($m[11]) << 8 ) + ord($m[12]), (ord($m[3]) << 8 ) +
my $l_a = new Math::Matrix[(ord($m[1]) << 8 ) + ord($m[2]), (ord($m[5]) << 8 ) + ord($m[6]), (ord($m[9]) << 8 ) + ord($m
my $temp = ord($m[13]);
    my $ti = (ord($m[14]) << 24) + (ord($m[15]) << 16) + (ord($m[16]) << 8) + ord($m[17]);
    return ($a_v, $l_a, $temp, $ti);
}
###################################
## Get Measures from @matched
sub GetMeasures {
    my @m = @{$_[0]};
    # my @offset_ang_calibrated = @{$_[1]};
    # my @offset_lin_calibrated = @{$_[2]};
    my $a_v =
new Math::Matrix([((ord($m[7]) << 8 )+ ord($m[8]))-$offset_ang_calibrated[0])*100/($offset_ang_calibrated[0]),
(((ord($m[11]) << 8 ) +
ord($m[12]))-$offset_ang_calibrated[1])*100/($offset_ang_calibrated[1]),
(((ord($m[3]) << 8 ) +
ord($m[4]))-$offset_ang_calibrated[2])*100/($offset_ang_calibrated[2])]);
    # my $l_a =
new Math::Matrix([-(ord($m[1])-$offset_lin_calibrated[0])*(2.5/8.3)/($offset_lin_calibrated[0]),
-(ord($m[3])-$offset_lin_calibrated[1])*(2.5/8.3)/($offset_lin_calibrated[1]),
```

```
-(ord($m[5]) - $offset_lin_calibrated[2])*(2.5/8.3)/($offset_lin_calibrated[2])]);
    my $l_a =
new Math::Matrix([((ord($m[1]) << 8 ) + ord($m[2])) - $offset_lin_calibrated[0])*(2.5/5)/($offset_lin_calibrated[0]),
(((ord($m[5]) << 8 ) + ord($m[6])) - $offset_lin_calibrated[1])*(2.5/5)/($offset_lin_calibrated[1]),
(((ord($m[9]) << 8 ) + ord($m[10])) - $offset_lin_calibrated[2])*(2.5/5)/($offset_lin_calibrated[2])]);
    my $temp = ord($m[13]);
    my $ti = (ord($m[14]) << 24) + (ord($m[15]) << 16) + (ord($m[16]) << 8) + ord($m[17]);
    # print sprintf("%x", (ord($m[11]) << 8)) + (ord($m[10]));
    # print sprintf("%x", (ord($m[10]) << 8) +   (ord($m[11]))        );
    return ($a_v, $l_a, $temp, $ti);
}
###################################
## Mtx B sub CalcMtxB {
    my $B_aux = new Math::Matrix([0,0,0], [0,0,0], [0,0,0]);
    my $a_v = $_[0];
    # line 0
    $B_aux->[0][0] = 0;
    $B_aux->[0][1] = -$a_v->[0][2]*0.010;
    $B_aux->[0][2] =   $a_v->[0][1]*0.010;

    # line 1
    $B_aux->[1][0] =   $a_v->[0][2]*0.010;
    $B_aux->[1][1] = 0;
    $B_aux->[1][2] = -$a_v->[0][0]*0.010;

    #line 2
    $B_aux->[2][0] = -$a_v->[0][1]*0.010;
    $B_aux->[2][1] =   $a_v->[0][0]*0.010;
    $B_aux->[2][2] = 0;

    return $B_aux;
} ###################################
## Mtx C
sub CalcMtxC {
    my $a_v          = $_[0];
    my $b             = $_[1];
    my $c             = $_[2];
    my $c_next       = new Math::Matrix([0,0,0],[0,0,0],[0,0,0]);
    my $identity    = new Math::Matrix([1,0,0],[0,1,0],[0,0,1]);
    my $sigma       = ($a_v->multiply_scalar(0.010))->absolute;
    my $sigma_rad    = $sigma*PI/180;
    if($sigma != 0){
        $c_next = $identity->add($b->multiply_scalar(sin($sigma_rad)/$sigma_rad));
        $c_next = $c_next->add( ($b->multiply($b))->multiply_scalar((1-cos($sigma_rad))/($sigma_rad*$sigma_rad)) );
        $c_next = $c->multiply($c_next);
    }
    else{
        $c_next = ( $identity->add($b) )->add( ($b->multiply($b))->multiply_scalar(0.5) );
        $c_next = $c->multiply($c_next);
    }
    return $c_next;
}
###################################
## Ag(t) = C(t)*Ab(t)
sub ProjAccel {
    my   $l_a   = $_[0];
    my   $c      = $_[1];
    # Hardware compensation canceling
    $l_a->[0][1] += 1;
    #convert g's to m/s^2
   $l_a = $l_a->multiply_scalar(9.8);
    #$l_a->print("l a:");
    return ( ($c->multiply($l_a->transpose()))->transpose() );
}
###################################
## string to hex
sub iso2hex($) {
    my $hex = '';
    for (my $i = 0;$i < length($_[0]);$i++) {
        my $ordno = ord substr($_[0], $i, 1);
        $hex .= sprintf("[%02x]", $ordno);
```

```
    }       $hex =~ s/ $//;;
    $hex;
}
```

8 Apêndice B

Programa do microcontrolador PIC18F458.

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// MAIN.C

#include "main.h"
//#include <htc.h>
// DEFINES
// GLOBAL VARIABLES
//int rec = 0;
//unsigned char ola = 0;
//unsigned char a;
//unsigned char aux;
//unsigned char buf;
//unsigned char ack = 0;
//unsigned char ack2 = 0;
//unsigned char conv_result = 0x00;
//unsigned long out;
//unsigned long time;
//unsigned char str[10];
unsigned char adc_buffer_full = 1;
unsigned char channel=0;
unsigned char str[10];
unsigned char measures[6][2];
//unsigned long measures_long[6];
unsigned char measures_char[7];
//unsigned long window_measures[6][WINDOW_SIZE];
//unsigned long filtered_measures[6];
//unsigned char window = 0;
//unsigned char j = 0;
unsigned char delta_10ms = 0;
//unsigned long measures_average[6];
//unsigned long n = 0;
unsigned long time = 0;
unsigned long time_aux = 0;
unsigned char i = 0;
unsigned char time_byte[4];
//unsigned long aux;
unsigned char ack = 1;
char nono = 0x35;
//eeprom unsigned char var1[256];
////////////////////////////////
//// MAIN///////////////////////
////////////////////////////////
void main(void)
{
/*
GIE = 1;
PEIE = 1;
SSPIE = 1;
*/
// Configurations
 disableInterrupts();
 initUsart();
 confTIMER0();
// confTIMER1();
// InitI2cMaster();
// InitI2cSlave();
 enableInterrupts();  ///////////////
 ENABLE_TIMER0_INT // TIMER.h
```

```
/*
time = 0x7778797A;
while(1){

 // unsigned long to 4 bytes
 for(i=0;i<4;i++){
  time_aux = time >> (i*8);
  time_byte[i] = (unsigned char)time_aux;
  putch(time_byte[i]);
 }
 putch('\n');

}
*/
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// I2C //
/////////
 while(1) // MAIN WHILE
 {
/*  // PART RELATED TO I2C (temperature)

  SSPCON1bits.SSPEN = 0;
  SSPCON1bits.SSPEN = 1;
  DelayMs(1);
  INTCONbits.GIE = 1;
  PIE1bits.SSPIE = 1;

  INTCON2bits.RBPU = 0; // PORTB PULL-UPS ON

  TRISBbits.TRISB1 = 0;
  TRISBbits.TRISB2 = 0;
  TRISBbits.TRISB3 = 0;
  PORTBbits.RB1 = 0;
  PORTBbits.RB2 = 0;
  PORTBbits.RB3 = 0;

  DelayMs(5);

  InitI2cMaster();


  while(1){
  StartI2c();      // Begin transmission
//  RepStartI2c();     // Begin transmission (repeat)
//  putch('S'); nl
//  DelayUs(25);
  WriteI2c(0x91);     // Slave Address
  while(SSPCON2bits.ACKSTAT); // Check for Acknoledge from slave 0:sent; 1:not sent
//    putch('A'); tab

   WaitIdleI2c();     // Wait here while Lines arent idle
   SSPCON2bits.RCEN=1;   // enables reception
//    DelayMs(2);
   DelayUs(100);
//    putch('T');
   ltoa(SSPBUF, str);  // Get the byte received and convert it to a string
   putstr(str); nl   // Print the string

   SSPBUF = 0x00;

   WaitIdleI2c();     // Wait here while Lines arent idle
   SSPCON2bits.ACKDT = 0;
   SSPCON2bits.ACKEN = 1;
//    DelayMs(2);

   RepStartI2c();     // Begin transmission (repeat)

//    ltoa(SSPBUF, str);  // Get the byte received and convert it to a string
//    putstr(str); nl   // Print the string
```

```
    SSPCON2bits.RCEN=1;  // enables reception
//    ltoa(SSPBUF, str);  // Get the byte received and convert it to a string
//    putstr(str); nl   // Print the string
//    DelayMs(2);
//    SSPCON2bits.RCEN=1;  // enables reception
    DelayMs(100);

    StopI2c();
   }
*/
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//    while(1);
//     PART RELATED TO ADC

    if(adc_buffer_full){

    // Save ADC value
//    measures_char[channel] = ADRESH; 8 bits conversion
    measures[channel][0] = ADRESH;
    measures[channel][1] = ADRESL;
    // ADC becomes empty
    adc_buffer_full = 0;


    // Show me the next channel !
    if(channel == 5){
     channel = 0;

    // SaveCurrentTemperature();
     if(delta_10ms){
      //ShowMeasures(); nl
      SendMeasures();

      delta_10ms = 0;
     }

     //else{
     // n++;
     //}

    }
    else{channel++;}
    if(channel == 0){  // start ? if YES then SAMPLE/HOLD
     SampleAndHold(10);
    }

    // Start conversion
    ADC(channel);
   }

   //REST OF THE PROGRAM ! (SPENDING TIME !)


 } // MAIN WHILE END
} // MAIN END
/////////////////////////////////////
//// INT ////////////////////////////
/////////////////////////////////////
void enableInterrupts(void)
{INTCONbits.GIE=1;}
void disableInterrupts(void)
{INTCONbits.GIE=0;}
// INTERRUPT SERVICE ROUTINE
//---------------------------------------------------------------------------
// High priority interrupt vector
#pragma code InterruptVectorHigh = 0x08
void
InterruptVectorHigh (void)
{
```

```
  _asm
    goto InterruptHandlerHigh //jump to interrupt routine
  _endasm
}
//-------------------------------------------------------------------------
// High priority interrupt routine
#pragma code
#pragma interrupt InterruptHandlerHigh
void
InterruptHandlerHigh ()
{
 INTCONbits.GIE = 0;


 // TIMER 0
 if (INTCONbits.TMR0IF){                  //check for TMR0 overflow
  delta_10ms = 1;
  time++;
  TMR0L=TIMER0_VALUE_LOW;
  TMR0L=TIMER0_VALUE_HIGH;
  INTCONbits.TMR0IF = 0;              //clear interrupt flag
    }


 // ADC
 if(PIR1bits.ADIF){
//   measures[channel][0] = ADRESH;
//   measures[channel][1] = ADRESL;

  adc_buffer_full = 1;
  PIR1bits.ADIF = 0;
 }
 /*
 // I2C
 if(PIR1bits.SSPIF){
//   btoa(SSPSTAT, str);
//   putstr(str); putch('\n');

  PIR1bits.SSPIF = 0;
 }
 */
 INTCONbits.GIE = 1;
}
//-------------------------------------------------------------------------
void ltoa(long int value, char *dest)
{
 long int val=value;
 char resto=0, i=0, j=0, aux=0;
 if(value < 0)
  val = -1 * val;
 if(val == 0){
  dest[0]=0x30;
  dest[1]=0;
  return;
 }
 for(i=0; val > 0; i++)
 {
  resto = val % 10;
  val   = val / 10;
  dest[i] = resto + 0x30;
 }
 if(value < 0)
 {
  dest[i++] = '-';
 }
 dest[i--]=0;
 for(j=0; j<i; j++, i--)
 {
  aux = dest[j];
  dest[j] = dest[i];
  dest[i] = aux;
```

```
 }
}
void inttoa(int value, char *dest)
{
 int val=value;
 char resto=0, i=0, j=0, aux=0;
 if(!value)
 {
  dest[0]='0';
  dest[1]='\0';
  return ;
 }
 if(value < 0)
  val = -1 * val;
 for(i=0; val > 0; i++)
 {
  resto = val % 10;
  val   = val / 10;
  dest[i] = resto + 0x30;
 }
 if(value < 0)
 {
  dest[i++] = '-';
 }
 dest[i--]=0;
 for(j=0; j<i; j++, i--)
 {
  aux = dest[j];
  dest[j] = dest[i];
  dest[i] = aux;
 }
}
/*
void Sample(void){
 TRISD = 0x00;
 PORTD = 0x00;
}
void Hold(void){
 TRISD = 0x00;
 PORTD = 0xFF;
}
*/
void SampleAndHold(unsigned int sample_time){
 unsigned int i=0;
 TRISD = 0x00;
 PORTD = 0x00;
 while(sample_time > i){i++;}
 TRISD = 0x00;
 PORTD = 0xFF;
}
void SendMeasures(void){
 char check_sum = 0;

 // header 4 bytes
 putch(0xEA); // Our stuff
 putch(0xBB); // IMU
 putch(0x00); // Message type
 putch(0x0B); // Payload length
/*
 // data 7 bytes // 8 bits conversion
 putch(measures_char[0]);
 putch(measures_char[1]);
 putch(measures_char[2]);
 putch(measures_char[3]);
 putch(measures_char[4]);
 putch(measures_char[5]);
 putch(measures_char[6]);
*/
 // 10 bits conversions
 putch(measures[0][0]);
 putch(measures[0][1]);
```

```
putch(measures[1][0]);
putch(measures[1][1]);

putch(measures[2][0]);
putch(measures[2][1]);

putch(measures[3][0]);
putch(measures[3][1]);

putch(measures[4][0]);
putch(measures[4][1]);

putch(measures[5][0]);
putch(measures[5][1]);

// temperature dummy
putch(0x00);

// unsigned long to 4 bytes
for(i=0;i<4;i++){
 time_aux = time >> (i*8);
 time_byte[3-i] = (unsigned char)time_aux;
}
// time 4 bytes (big endien)
putch(time_byte[0]);
putch(time_byte[1]);
putch(time_byte[2]);
putch(time_byte[3]);

check_sum =
0xEA +
0xBB +
0x0B +
/* 8 bits conversion
measures_char[0]+
measures_char[1]+
measures_char[2]+
measures_char[3]+
measures_char[4]+
measures_char[5]+
measures_char[6]+
*/

// 10 bits conversion
measures[0][0]+
measures[0][1]+

measures[1][0]+
measures[1][1]+

measures[2][0]+
measures[2][1]+

measures[3][0]+
measures[3][1]+

measures[4][0]+
measures[4][1]+

measures[5][0]+
measures[5][1]+


time_byte[0]+
time_byte[1]+
time_byte[2]+
time_byte[3];


//check sum 1 byte
```

```
 putch(check_sum);

}
/*
unsigned long DoubleByteToLong(unsigned char high, unsigned char low){
 unsigned long aux=0;
 unsigned long high_aux;
 unsigned long low_aux;
 low_aux = (unsigned long)low;
 high_aux = (unsigned long)high;
 high_aux = high_aux << 8;
 aux = high_aux | low_aux;
 return aux;
}
*/
void ShowMeasures(void){
 char i;
 for (i=0;i<6;i++){
  ltoa(measures_char[i], str);
  putstr(str); tab
 }
}
/*
void ShowMeasuresAverage(void){
 char i;
 for (i=0;i<6;i++){
  ltoa(measures_average[i], str);
  putstr(str); tab
 }
}
*/
/*
void FilterMeasures(void){
 if(window < WINDOW_SIZE){
  window_measures[0][window] = measures_long[0];
  window_measures[1][window] = measures_long[1];
  window_measures[2][window] = measures_long[2];
  window_measures[3][window] = measures_long[3];
  window_measures[4][window] = measures_long[4];
  window_measures[5][window] = measures_long[5];

  filtered_measures[0] += window_measures[0][window];
  filtered_measures[0] += window_measures[1][window];
  filtered_measures[0] += window_measures[2][window];
  filtered_measures[0] += window_measures[3][window];
  filtered_measures[0] += window_measures[4][window];
  filtered_measures[0] += window_measures[5][window];


  window++;
 }
 else{
  filtered_measures[0] += measures_long[0];
  filtered_measures[0] -= window_measures[0][j];


  filtered_measures[1] += window_measures[1][j];
  filtered_measures[2] += window_measures[2][j];
  filtered_measures[3] += window_measures[3][j];
  filtered_measures[4] += window_measures[4][j];
  filtered_measures[5] += window_measures[5][j];

  window_measures[0][j] = measures_long[0];
  window_measures[1][j] = measures_long[1];
  window_measures[2][j] = measures_long[2];
  window_measures[3][j] = measures_long[3];
  window_measures[4][j] = measures_long[4];
  window_measures[5][j] = measures_long[5];

  j++;
  if(j > 9){j = 0;}
```

```
 }

}
*/

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// ADC.C

#include "ADC.h"
void ADC(unsigned char channel){
unsigned char i;
i = 0;

// A/D RESULT JUSTIFICATION
 RIGHT_JUSTIFIED
// PORT CONFIGURATION CONTROL
 ALL_ANALOGIC
// CONVERSION CLOCK FREQUENCY
 FREQ_OSC_32    // For XTAL = 20MHz
// CHANNEL
    if(channel == 0){ CHANNEL_0 } // pin2 (RA0)
  else if(channel == 1){ CHANNEL_1 } // pin3 (RA1)
  else if(channel == 2){ CHANNEL_2 } // pin4 (RA2)
  else if(channel == 3){ CHANNEL_3 } // pin5 (RA3)
  else if(channel == 4){ CHANNEL_4 } // pin7 (RA5)
  else if(channel == 5){ CHANNEL_5 } // pin8 (RE0)
  else if(channel == 6){ CHANNEL_6 } // pin9 (RE1)
  else if(channel == 7){ CHANNEL_7 } // pin10(RE2)
// ADC SWITCH ON/OFF
 ADC_ON
// INTERRUPTS
 INTERRUPTS_ON
// DELAY AQUISITION TIME 13us
 INSTRUCTION_64
// CONVERSION SWITCH    ... Start conversion
 CONVERSION_ON
//            ... interrupt !
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// I2C.C

#include "I2C.h"
// INIT MASTER
void InitI2cMaster(void){
 TRISCbits.TRISC3  = 1;
 TRISCbits.TRISC4  = 1;
 SSPCON1  = 0x38;
 SSPCON2 = 0x00;
// SSPSTAT = 0x80;
 SSPSTATbits.SMP = 1;
 SSPSTATbits.CKE = 1;

 SSPADD  = 49; // 20M/(4*SSPADD+1) = 100K <=> SSPADD = 49
// SSPADD  = 0x0C; // 400kHz
 PIR1bits.PSPIF   = 0; // Parallel Slave Port RESERVED on PIC16F873/876; MUST BE CLEARED
 PIR2bits.BCLIF   = 0; // Bus collision Detection cleared
 PIE1bits.SSPIE   = 1; // Synchronous Serial Port Interrupt Bit enabled
 INTCONbits.PEIE    = 1; // Peripheral Interrupt Enable Bit enabled
}
// INIT SLAVE
void InitI2cSlave(void){
 TRISCbits.TRISC3  = 1;
 TRISCbits.TRISC4  = 1;
 SSPCON1  = 0x36;
 SSPCON2 = 0x00;
 SSPSTAT = 0x80;
 SSPADD  = 2;

 PIR1bits.PSPIF   = 0; // Parallel Slave Port RESERVED on PIC16F873/876; MUST BE CLEARED
 PIR2bits.BCLIF   = 0; // Bus collision Detection cleared
```

```
 SSPCON2bits.GCEN    = 1;  // General Call Enabled
 PIE1bits.SSPIE   = 1; // Synchronous Serial Port Interrupt Bit enabled
 INTCONbits.PEIE    = 1; // Peripheral Interrupt Enable Bit enabled
}
// START
void StartI2c(void){
 WaitIdleI2c();
 SSPCON2bits.SEN=1;  // automatically cleared by hardware
}
// REPEATED START
void RepStartI2c(void){
 WaitIdleI2c();
 SSPCON2bits.RSEN=1;  // automatically cleared by hardware
}
// STOP
void StopI2c(void){
 WaitIdleI2c();
 SSPCON2bits.PEN=1;  // automatically cleared by hardware
}
// READ SLAVE
unsigned char ReadSlaveI2c(void){
 unsigned char i2cReadData;
 WaitIdleI2c();
// RCEN=1;
// ShowRegister(SSPBUF); nl
// ShowI2cRegisters();
// WaitIdleI2c();
// delay(1000);
 i2cReadData = SSPBUF;
// WaitIdleI2c();
// ACKEN = 1; // send acknowledge
/*
 if ( ack )
  {
   ACKDT=0;
  }
 else
  {
   ACKDT=1;
  }
  ACKEN=1;                // send acknowledge sequence
*/
 return( i2cReadData );
}
// READ MASTER
unsigned char ReadMasterI2c(void){
 unsigned char i2cReadData;
 WaitIdleI2c();
 SSPCON2bits.RCEN=1;
// ShowRegister(SSPBUF); nl
// ShowI2cRegisters();
// WaitIdleI2c();
 delay(1000);
 i2cReadData = SSPBUF;
 WaitIdleI2c();
 SSPCON2bits.ACKEN = 1; // send acknowledge
/*
 if ( ack )
  {
   ACKDT=0;
  }
 else
  {
   ACKDT=1;
  }
  ACKEN=1;                // send acknowledge sequence
*/
 return( i2cReadData );
}
// WRITE
unsigned char WriteI2c(unsigned char i2cWriteData){
```

```
 WaitIdleI2c();
 SSPBUF = i2cWriteData;
// delay(35); // Delay must be higher than 31 to allow ACKNOWLEDGE reception
// return ( ! ACKSTAT  ); // function returns '1' if transmission is acknowledged
 return 0;
}
// WAIT FOR IDLE
void WaitIdleI2c(){
 while (( SSPCON2 & 0x1F ) | SSPSTATbits.R_W ) {}; // wait for idle and not writing
}
void ShowI2cRegisters(void){
 unsigned char string[9];
 putch('S');
 btoa(SSPSTAT, string);
 putstr(string); tab
 putch('C');
 btoa(SSPCON1, string);
 putstr(string); tab
 putch('C');putch('2');
 btoa(SSPCON2, string);
 putstr(string);
}
void ShowRegister(unsigned char reg){
 unsigned char bits[9];
 btoa(reg, bits);
 putstr(bits);
}
void btoa(unsigned char val, unsigned char *str){
 str[0] = ( (val & 0x80) >> 7) + 0x30;
 str[1] = ( (val & 0x40) >> 6) + 0x30;
 str[2] = ( (val & 0x20) >> 5) + 0x30;
 str[3] = ( (val & 0x10) >> 4) + 0x30;
 str[4] = ( (val & 0x08) >> 3) + 0x30;
 str[5] = ( (val & 0x04) >> 2) + 0x30;
 str[6] = ( (val & 0x02) >> 1) + 0x30;
 str[7] = (val & 0x01) + 0x30;
 str[8] = 0;
}
// DELAY ALPHA
void delay(unsigned int count){
unsigned int i;
 for (i=0;i<count;++i){}
}
// INTERRUPT SERVICE ROUTINE
// MASTER TRANSMISSION ROUTINE
/*
void I2cInterruptServiceRoutine(void){
 static char state=0;
 if (state == 0){
  state=1;
  WriteI2c(0x02);
 }
 if ((state == 1) && (ACKSTAT == 0)){
  state=2;
  WriteI2c(0x05);
 }
 if((state == 2) && (ACKSTAT == 0)){
  state=0;
  StopI2c();
 }
}
*/
// SLAVE RECEPTION ROUTINE
void I2cInterruptServiceRoutine(void){
 static char state=0;
 unsigned char buffy;
 if((state == 0)){
  state = 1;
  buffy = SSPBUF;
  ShowRegister(buffy); nl
 }
```

```
 else{
  putch('F');
 }
 if(state == 1){
  state = 0;
  buffy = SSPBUF;
  ShowRegister(buffy); nl
 }
 else{
  putch('F');
 }
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// DELAY.C

/*
 * Delay functions
 * See delay.h for details
 *
 * Make sure this code is compiled with full optimization!!!
 */
#include "delay.h"
void
DelayMs(unsigned char cnt)
{
#if XTAL_FREQ <= 2MHZ
 do {
  DelayUs(996);
 } while(--cnt);
#endif
#if    XTAL_FREQ > 2MHZ
 unsigned char i;
 do {
  i = 4;
  do {
   DelayUs(250);
  } while(--i);
 } while(--cnt);
#endif
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// USART.C


/*
 * USART MODULE
 *
 * files: usart.c, usart.h
 *
 */
#include "usart.h"
void initUsart()
{
 TRISCbits.TRISC7 = 1; // RX (input)
 TRISCbits.TRISC6 = 0; // TX (output)
 SPBRG = 10;  //baud 10->115200; 21->57600
 TXSTAbits.SYNC = 0;
 RCSTAbits.SPEN = 1;
 TXSTAbits.BRGH = 1;
 RCSTAbits.CREN = 1;
 INTCONbits.PEIE = 0; // Periferal interrupt enable bit (inturrupções p/ usart)
 PIE1bits.RCIE = 0;
 PIE1bits.TXIE = 0;
 TXSTAbits.TXEN = 1;
}
void putstr(char *str)
{
 while((*str)>0)
  putch( *(str++) );
```

```
}
void putch(unsigned char byte)
{
 while(!PIR1bits.TXIF)
  continue;
 TXREG = byte;
}
unsigned char getch()
{
 while(!PIR1bits.RCIF)
  continue;
 return RCREG;
}
void SerString_rom(static char rom *str) // send string via RS232
{
 while((*str)>0)
  putch( *(str++) );
}
void putch_uchar ( unsigned char value )
{
 unsigned char i;
 char numero[10];
 if (value == 0) {
  putch('0');
  return;
 }
 i=0;
 while (value) {
  //putch ( '0' + (value % 10) );
  numero[i++] =  '0' + (value%10);
  value = value / 10;
 }
 while(i--)
  putch(numero[i]);
}
void putch_ushortlong( unsigned short long value)
{
 unsigned char i;
 char numero[10];
 if (value == 0) {
  putch('0');
  return;
 }

 i=0;
 while (value) {
  numero[i++] =  '0' + (value%10);
  value = value / 10;
 }
 while(i--)
  putch(numero[i]);
}
unsigned char getch_number()
{
 //insira um numero
 unsigned char value=0, tmp=0;

 putch('\n');
 putch('\r');
 putch(':');
 tmp = getch();
 while (tmp != 13)
 {
  if ( (tmp >= '0') && (tmp <='9') ) value = value*10 + (tmp-'0');
  if (tmp == 8) value = value/10;
  putch('\r');
  putch(':');
  putch_uchar(value);
  putch(' ');
  putch(' ');
  putch(' ');
```

```
  tmp = getch();
 }
 putch('\r');
 putch(':');
 putch_uchar(value);
 putch(' ');
 putch(' ');
 putch(' ');
 return value;
}
void putch_ulong( unsigned long value )
{
 unsigned char i;
 char numero[10];
 if (value == 0) {
  putch('0');
  return;
 }

 i=0;
 while (value) {
  numero[i++] =  '0' + (value%10);
  value = value / 10;
 }
 while(i--)
  putch(numero[i]);
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// TIMER0.C

/*
 * TIMER0 MODULE
 *
 * files: TIMER0.c, TIMER0.h
 *
 */

#ifndef __TIMER0_C__
#define __TIMER0_C__
#include "TIMER0.h"
// Global Variables
// Functions
void confTIMER0(void)
{
// OPTION REGISTER CONFIGURATION
 T0CONbits.T0CS = OPERATION_TYPE;
 T0CONbits.T0SE = LOW_TO_HIGH;
 T0CONbits.PSA = 1;    // PRESCALER: 1=not assigned; 0=assigned
 T0CONbits.T08BIT = 0; // 8bits = 1; 16bits = 0
 switch (TIMER0_PRESCALER)
 {
  case 2:   T0CONbits.T0PS2=0; T0CONbits.T0PS1=0; T0CONbits.T0PS0=0; break;
  case 4:   T0CONbits.T0PS2=0; T0CONbits.T0PS1=0; T0CONbits.T0PS0=1; break;
  case 8:   T0CONbits.T0PS2=0; T0CONbits.T0PS1=1; T0CONbits.T0PS0=0; break;
  case 16:  T0CONbits.T0PS2=0; T0CONbits.T0PS1=1; T0CONbits.T0PS0=1; break;
  case 32:  T0CONbits.T0PS2=1; T0CONbits.T0PS1=0; T0CONbits.T0PS0=0; break;
  case 64:  T0CONbits.T0PS2=1; T0CONbits.T0PS1=0; T0CONbits.T0PS0=1; break;
  case 128: T0CONbits.T0PS2=1; T0CONbits.T0PS1=1; T0CONbits.T0PS0=0; break;
  case 256: T0CONbits.T0PS2=1; T0CONbits.T0PS1=1; T0CONbits.T0PS0=1; break;
  default:  T0CONbits.T0PS2=1; T0CONbits.T0PS1=1; T0CONbits.T0PS0=1; break;
 }
 TMR0L=TIMER0_VALUE_LOW; //230; // 100
 TMR0H=TIMER0_VALUE_HIGH;
// INTCONbits.T0IE=0;// TIMER0 interrupt enabled
}
#endif
```

Referências

[1] Telmo Reis Cunha. High Precision Navigation Integrating Satellite Information - GPS - and Inertial System Data. Tese de Doutoramento, Faculdade de Engenharia da Universidade do Porto, 2002

[2] Oliver J. Woodman. An Introduction to inertial navigation. University of Cambridge, 2007

[3] A. D. King. Inertial Navigation - Forty Years of Evolution. 1998

[4] Chen, W. and P.A. Cross (1990) "Integration of GPS and Inertial System for Precise Surveying Applications." Department of Surveying, University of Newcastle upon Tyne, pp. 375 - 395.

[5] Nayak, R.A. (2000) "Reliable and Continuous Urban Navigation Using Multiple GPS Antennas and a Low Cost IMU." Master Thesis, Department of Geomatics Engineering, University of Calgary, Alberta.

[6] Shin, E.H. (2001) "Accuracy Improvement of Low Cost INS/GPS for Land Applications." Master Thesis, Department of Geomatics Engineering, University of Calgary, Alberta.

[7] Schwarz, K.P., H.E. Mantell, N. EI-Sheimy, R. Li, M. A. Chapman and D. Cosandier 1993: VISAT - A Mobile Highway Survey System of High Accuracy. Proceedings of the VNIS'93, pp.476-481. Stevens

[8] Magno Morgado, G., "Desenvolvimento e análise de desempenho de um sistema de navegação integrado", dissertação de Mestrado Integrado em Engenharia Electrónica e Telecomunicações, Universidade de Aveiro, Nov. 2008.